IBM

Reference Manual

FOR TRANSIT Automatic Coding System for the IBM 650 Data Processing System

IBM Reference Manual

FOR TRANSIT Automatic Coding System

for the IBM 650 Data Processing System

PREFACE

FOR TRANSIT is an automatic coding system for the IBM 650 Data Processing System. By superimposing a translator on the compiler* developed at the Carnegie Institute of Technology, FOR TRANSIT makes available to 650 users the language of FORTRAN. FORTRAN, a language developed for the IBM 704, closely resembles the language of mathematics; it was designed primarily for scientific and engineering computation. FOR TRANSIT in effect transforms the 650 into a machine with which communication can be made in a language more concise and more familiar than the 650 language itself. The result should be a considerable reduction in the training required to program, as well as in the time consumed in writing programs and eliminating programming errors.

This manual will serve as a reference for both FOR TRANSIT I and FOR TRANSIT II inasmuch as the language and procedures of the two versions are essentially the same. FOR TRANSIT I is designed for the basic machine and FOR TRANSIT II for the 650 equipped with indexing registers and automatic floating decimal arithmetic. Both versions of the system were revised and re-issued late in 1958 and this reference manual reflects the changes made in the system.

---

* Internal Translator (IT) A Compiler for the 650, by A. J. Perlis, J. W. Smith, and H. R. Van Zoeren, Computation Center, Carnegie Institute of Technology. (See IBM 650 Library Program Abstract 2.1.001.)

TABLE OF CONTENTS

INTRODUCTION

A FOR TRANSIT program consists of a sequence of FORTRAN statements. The following brief program will serve to illustrate the general appearance and some of the properties of a FOR TRANSIT program.

Example of a FOR TRANSIT Program

C ◄ FOR COMMENT	STATEMENT NUMBER 1 5	CONTINUATION 6	FORTRAN STATEMENT 7 72
C			PROGRAM FOR FINDING THE LARGEST VALUE
C		X	ATTAINED BY A SET OF NUMBERS
			BIGA = A(I)
			DO 20 I = 2, N
			IF (BIGA - A(I)) 10, 20, 20
	10		BIGA = A(I)
	20		CONTINUE

This program examines the set of n numbers a_i($i = 1, \ldots, n$) and sets the quantity BIGA to the largest value attained in the set. It begins (after a comment describing the program) by setting BIGA equal to a_1. Next the DO statement causes the succeeding statements to and including statement 20 to be carried out repeatedly, first with $i = 2$, then with $i = 3$, etc., and finally with $i = n$. During each repetition of this loop the IF statement compares BIGA with a_i; if BIGA is less than a_i, statement 10, which replaces BIGA by a_i, is executed before continuing.

Description of the System

The FOR TRANSIT system consists of three major parts:

1. The translator, FOR TRANSIT, which accepts FORTRAN statements and produces corresponding IT statements.

2. The compiler, a modification of IT, which accepts IT statements and compiles 650 instructions in symbolic (SOAP II) language.

3. The assembler, a modified version of SOAP II*, which produces an optimized machine language program from the symbolic instructions.

---

* SOAP II, Symbolic Optimal Assembly Program for the IBM 650 Data Processing System. (See SOAP II Programmer's Reference Manual, Form C28-4000.)

Each of these parts requires a pass on the 650 . The passes are referred to as translation phase, compilation phase, and assembly phase, or more simply as phases I, II, and III. The programmer need concern himself only with the FORTRAN language since the translation, compilation, and assembly phases are completely automatic. The FORTRAN language program is referred to as the source program; the 650 language program, which is available at the completion of phase III, is referred to as the object program. A schematic representation of the over-all system is shown in Figure 1.

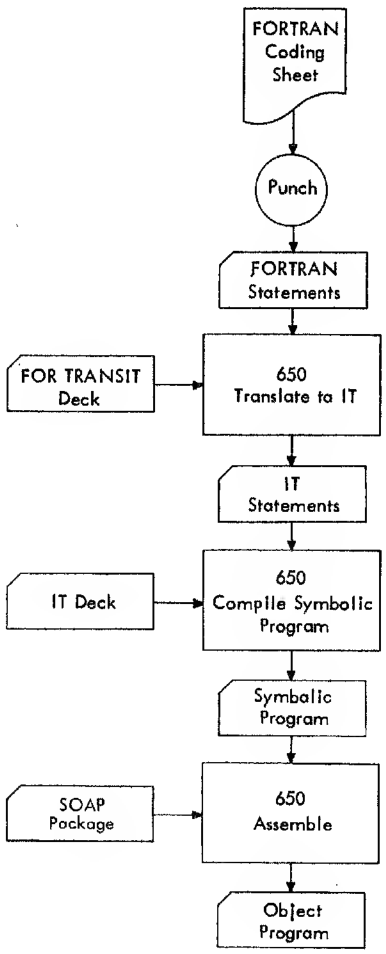


Figure 1

Phases I and II of the FOR TRANSIT system are designed to permit multiple processing of source programs. By translating several source programs while the FOR TRANSIT program is in memory, then compiling the same programs on the second pass, considerable savings of machine time and card handling may be effected.

Machine Requirements

The FOR TRANSIT system is presently available in two versions which are designated FOR TRANSIT I and FOR TRANSIT II. In order to make the system as widely applicable as possible, each of the two versions is provided in two forms corresponding to different equipment specifications as indicated in the following paragraphs.

FOR TRANSIT I, which produces object programs for the basic 650, is available in these forms:

1. The standard FOR TRANSIT I is designed to run on a 650 equipped as indicated below. Note that the special character device is not required even though the input cards for the system contain special characters.

   2000 word drum
   Alphabetic device
   20 pilot selectors (10 are standard)
   16 co-selectors (8 are standard)
   10 read code and 10 punch code selectors
   1 half-time emitter (read side)

2. FOR TRANSIT I (S) is provided for those installations in which the 650 is equipped with a special character device. The specific equipment requirements are as follows:

   2000 word drum
   Alphabetic device
   Special character device (Group II)

FOR TRANSIT II produces object programs designed to be run on the IBM 650 equipped with indexing registers and automatic floating decimal arithmetic. These additional features are not required, however, on the processor, i.e., the 650 on which the object program is compiled. FOR TRANSIT II, as well as I, is available in two forms to provide for different configurations of equipment. FOR TRANSIT II runs on the same machine as specified for FOR TRANSIT I, and the machine requirements for running FOR TRANSIT II(S) are identical to those for FOR TRANSIT I(S).

Program Decks

The several program decks for the FOR TRANSIT system may be obtained from IBM Applied Programming Publications. Requests should specify the version desired, i.e., FOR TRANSIT I, I(S), II, or II(S), and should be addressed to:

IBM 650 Program Librarian
Applied Programming
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

In order to facilitate the filling of requests, the various card decks in the system deck packages will not be labeled individually, but can be identified by the high order digit end-printed on the cards as indicated below:

Deck No.	Deck Name
1	FOR TRANSIT deck
2	IT (Compiler) deck
3	SOAP-PACKAGE deck

The cards in each deck will be end-printed serially (in the three low order positions) beginning with 001.

Outline of Manual

The FOR TRANSIT system will be presented in four sections. A glossary of some of the terms peculiar to the computing vernacular is included for reference in Appendix C. The manual proceeds as follows:

1. How to write the source program.

   This includes a definition of constants, variables, and subscripts; the types of FORTRAN statements are described and examples given of each.

2. Incorporating of subroutines.

   A subroutine is a program that is integrated into a larger program. Subroutines are used for such things as the evaluation of a function, e.g., square root. A number of subroutines are included in the system, e.g., computing integral exponentials, reading in data and punching out results. In this chapter the built-in subroutines as well as rules for incorporating subroutines not included in the system are discussed.

3. Processing the source program.

   The preparation of statement cards is covered, as well as operating instructions for phases I, II, and III.

4. Using the object program.

   Card format for data used by the object program as well as the actual executing of the object program is included in the final chapter.

5. Appendixes.

   Wiring diagrams, listings of sample cards and a glossary of terms are included in the Appendixes.

Consider the algebraic formula for one of the two roots of a quadratic equation:

$$\text{Root} = \left[- B + \sqrt{B^2 - 4AC}\right] /2A$$

The FORTRAN language statement which creates a machine language program for this calculation is as follows:

```
Root = (-B +SQRTF(B**2 - 4.0*A*C)) / (2.0*A)
```

where:

1. The meaning of the statement is: evaluate the expression on the right side of the equal sign and make this the value of the variable on the left side of the equal sign.

2. The symbol * denotes multiplication.

3. The symbol ** denotes exponentiation, e.g., A**3 means A^3.

4. SQRTF (arg) is a subroutine which computes the square root of the argument enclosed in parentheses.

Source programs for the FOR TRANSIT system consist of a sequence of FORTRAN language statements. FORTRAN statements may be grouped into four classifications: arithmetic statements, control statements, input/output statements, and specification statements. The statement above for the root is an example of an arithmetic statement. The following list includes all of the FORTRAN statements which are permissible in the FOR TRANSIT system:

A. Arithmetic statements

   e.g., a = b

B. Control statements

   1. GO to n
   2. GO TO (n_1, n_2, . . ., n_m), i
   3. IF (a) n_1, n_2, n_3

4. PAUSE
5. STOP
6. DO n i = m_1, m_2
   DO n i = m_1, m_2, m_3
7. CONTINUE
8. END

C. Input/output statements

1. READ n, list
2. PUNCH n, list

D. Specification statements

1. DIMENSION V, V, V, ...
2. EQUIVALENCE (a, b, c), (d, e)....

The FORTRAN system was designed for a more complex machine than the 650, and consequently some of the 32 statements found in the FORTRAN Programmer's Reference Manual are not acceptable to the FOR TRANSIT system. In addition, certain restrictions to the FORTRAN language have been added. However, none of these restrictions make a source program written for FOR TRANSIT incompatible with the FORTRAN system for the IBM 704. For example, whereas FORTRAN variables may consist of from one to six characters in the FORTRAN system, FOR TRANSIT requires that variables consist of from one to five characters. It should be noted that in a few instances FORTRAN restrictions have been relaxed to take advantage of certain features of the 650; specific information regarding such modifications is included at the applicable places in the following pages for the benefit of users concerned with compatibility.

The following pages are devoted to a formulation of the rules for constructing FORTRAN statements and for using them to write a FORTRAN language program. The last part of this chapter includes some brief examples of FORTRAN programs with explanatory notes.

Statements and Statement Numbers

Statements

Each statement is punched in one or more cards. Maximum statement length is 125 characters, exclusive of blanks. (For statement and comments cards formats and punching details, see Chapter III.)

Statement Numbers

The order of statements is governed solely by the order of the cards. However, cross-referencing within a program may be accomplished by assigning statement numbers to those statements referred to. Any unsigned fixed point constant from 0001 to 0999 may be used as a statement number.

Statements need not be in numerical order nor do all statements need statement numbers. In FOR TRANSIT I and II unnumbered statements are preceded by zeros since the statement number field of the card must contain numerical punches. In FOR TRANSIT I (S) and II (S) the statement number field may be left blank.

Constants, Variables, and Subscripts

FORTRAN statements may refer to constants, variables or entire arrays of numbers.

Constants

Two types of constants are permissible: fixed point (restricted to integers), and floating point (characterized by being written with a decimal point).

Fixed Point Constants

GENERAL FORM	EXAMPLES
1 to 10 decimal digits. A preceding + or - sign is optional.	3 +1 -28987

NOTE: The magnitude of fixed point constants in the FORTRAN system for the IBM 704 must be less than 32768. FOR TRANSIT users must comply with this restriction if compatibility is desired.

Floating Point Constants

GENERAL FORM	EXAMPLES
1 to 8 decimal digits, with a decimal point at the beginning, at the end, or between two digits. A preceding + or - sign is optional. A decimal exponent (a 1- or 2-digit fixed point constant) preceded by an E may follow.	17. 5.0 -.0003 5.0E3 (=5.0 x 10^3) 5.9E+3 (=5.9 x 10^3) 5.0E-7 (=5.0 x 10^{-7}) 5.3E13 (=5.3 x 10^{13})

The number will appear in the object program as a normalized single-precision floating point number of the form .xxxxxxxx PP where PP is the power of 10 with 50 added to avoid negative exponents. If the number is written with more than eight decimal digits, it will be truncated to eight digits.

NOTE: The magnitude of floating point constants in the FORTRAN system for the 704 must lie between the approximate limits of 10^{-38} and 10^{38}. FOR TRANSIT users must comply with this restriction if compatibility is desired. The magnitude limits of floating point constants in the FOR TRANSIT system are 10^{-49} and 10^{50}.

Variables

Two types of variables are also permissible: fixed point (restricted to integral values) and floating point. Fixed point variables are distinguished by the fact that their first character is I, J, K, L, M, or N.

Fixed Point Variables

GENERAL FORM	EXAMPLES
1 to 5 alphabetic or numerical characters (not special characters) of which the first is I, J, K, L, M, or N.	I M2 JOBNO

Floating Point Variables

GENERAL FORM	EXAMPLES
1 to 5 alphabetic or numerical characters (not special characters) of which the first is alphabetic but not I, J, K, L, M, or N.	A B7 DELTA

NOTE: 1. The name of a variable must not be the same as the name of any function used in the program after the terminal F of the function name has been removed, nor should the name of a subscripted variable end with an F.

2. A maximum of 100 variables of which 20 may be subscripted may be used in any one program.

Subscripts and Subscripted Variables

A variable can be made to represent any member of a 1- or 2-dimensional array of quantities by appending to it 1 or 2 subscripts; the variable is then a subscripted variable. The subscripts are fixed point quantities whose values determine which member of the array is being referred to.

Subscripts

GENERAL FORM	EXAMPLES
Let v represent any fixed point variable and c (or c') any unsigned fixed point constant. Then a subscript is an expression of one of the forms v c v+c or v-c c*v c*v+c' or c*v-c'	I 3 MU + 2 MU - 2 5 * J 5 * J + 2 5 * J - 2

The variable v must not itself be subscripted.

Subscripted Variables

GENERAL FORM	EXAMPLES
A fixed or floating point variable followed by parentheses enclosing 1 or 2 subscripts separated by commas.	A (I) K (3) BETA (5*J-2, K+2)

For each variable that appears in subscripted form, the size of the array, i.e., the maximum values which its subscripts can attain, must be stated in a DIMENSION statement preceding the first appearance of the variable.

The minimum value which a subscript may assume in the object program is +1.

NOTE: A 2-dimensional array A will, in the object program, be stored sequentially in the order $A_{1,1}$, $A_{2,1}$, . . . , $A_{m,1}$, $A_{1,2}$, $A_{2,2}$, . . . , $A_{m,2}$, . . . , $A_{m,n}$. Thus it is stored "columnwise", with the first of its subscripts varying more rapidly. 1-dimensional arrays are of course simply stored sequentially.

Functions, Expressions, and Arithmetic Formulas

Of the various FORTRAN statements, it is the arithmetic formula which defines a numerical calculation that the object program is to do. A FORTRAN arithmetic formula resembles very closely a conventional arithmetic formula; it consists of the variable to be computed, followed by an " = " sign, followed by an arithmetic expression. For example, the arithmetic formula

```
Y = A - SINF (B-C)
```

means "replace the value of y by the value of a-sin (b-c)."

Functions

As in the above example, a FORTRAN expression may include the name of a function (e.g., the sine function SINF), provided the routine for evaluating the function is available to the FOR TRANSIT system.

GENERAL FORM	EXAMPLES

The name of the function is 4 or 5 alphabetic or numerical characters (not special characters), of which the last must be F and the first must be alphabetic. Also, the first must be X if and only if the value of the function is to be fixed point. The name of the function is followed by parentheses enclosing the argument (which may be expressions).	SINF(A+B)
	SQRTF(SINF(A))
	XABSF(3.*X)
	SAMPF(A, B, C)

The FOR TRANSIT system has some built-in subroutines for evaluating functions. However, provision is made for the user to incorporate up to ten subroutines in any one program. Detailed information concerning the inclusion of subroutines is given in Chapter II. For the purposes of this section it will suffice to indicate that by means of function title cards prepared by the user, an internal table of function subroutines is created in the FOR TRANSIT system. When a source program is being processed, any function encountered will be checked against the table, and an appropriate entry will be generated by the system.

The X character in function names is meaningless in the FOR TRANSIT system but is specified for compatibility with

704 FORTRAN. Compatibility also requires that any function name used in a source program represent a subroutine available to the FORTRAN system.

Expressions

An expression is any sequence of constants, variables (subscripted or non-subscripted), and functions, separated by operation symbols, commas, and parentheses so as to form a meaningful mathematical expression. The mode of arithmetic in expressions may be either floating or fixed point. When the mode is mixed, i.e., the expression includes both floating and fixed point variables, the arithmetic will be performed in the floating point mode. FOR TRANSIT users concerned with compatibility of programs with the 704 FORTRAN system are cautioned regarding mixed expressions; the mixing of modes is permitted in the FORTRAN system only under certain conditions as noted in the FORTRAN manual.

Brief rules for forming expressions follow.

1. The five basic operations of the system are specified by the symbols +, -, *, /, and **, which denote addition, subtraction, multiplication, division, and exponentiation, respectively.

2. Two operation symbols may not appear in sequence. Thus A* - B is not a valid expression, but A*(-B) is valid.

3. When the hierarchy of operations in an expression is not completely specified by parentheses, the order of operations (working from inside to outside) is assumed to be exponentiation, then multiplication and division, and finally addition and subtraction. Thus the expression A +B/C +D**E*F - G will be taken to mean $A+(B/C)+(D^{E}\cdot F)-G$.

4. When the sequence of consecutive operations of the same hierarchal level (i.e., consecutive multiplications and divisions) is not completely specified by parentheses, the order of operations is assumed to be from left to right for floating point variables, and from right to left for fixed point variables. For instance, the expression A*B/C*D is taken to mean ((A*B)/C)*D; and the expression I*J/K*L is taken to mean I*(J/(K*L)).

5. The expression $A^{B^{C}}$, which is sometimes considered meaningful, cannot be written as A**B**C; it should be written as (A**B)**C or A**(B**C), whichever is intended.

The following are some examples of correct and incorrect ways of forming expressions in FORTRAN language.

The expression	should not be written as	but should be written
A/-B	A/-B	A/(-B)
AB or A · B	AB	A*B
A^{I+2}	A**I +2	A**(I+ 2)
$A^{I+2} \cdot B$	A**I + 2*B	A**(I+2)*B
$\frac{AB}{CD}$	A*B/C*D	(A*B) / (C*D) or A*B / (C*D) or A/C * B/D

Modes of Arithmetic in Exponentiation

FOR TRANSIT has built-in provision for handling exponentiation as follows:

1. A plus or minus fixed point quantity or a plus or minus floating point quantity may be raised to a power which is a plus or minus fixed point quantity.

2. A plus or minus floating point quantity may be raised to a power which is a plus or minus floating point quantity.

   NOTE: This operation will give the absolute value of the quantity raised to the plus or minus power.

Exponentiation conforming to these specifications is handled by subroutines contained in the SOAP - PACKAGE deck and thus is actually performed at object program time.

Verification of Correct Use of Parentheses

The following procedure is suggested for checking that the parentheses in a complicated expression correctly express the desired operations.

Label the first open parenthesis "1"; thereafter, working from left to right, increase the label by 1 for each open parenthesis and decrease it by 1 for each closed parenthesis. The label of the last parenthesis should be 0; the mate of an open parenthesis labeled n will be the next parenthesis labeled n - 1.

FOR TRANSIT permits a maximum nest of nine parentheses.

Arithmetic Formulas

GENERAL FORM	EXAMPLES
"a = b" where a is a variable (subscripted or not subscripted) and b is an expression.	A(I) = B(I)+SINF(C (I))

The " = " sign in an arithmetic formula has the meaning "is to be replaced by." An arithmetic formula is therefore a command to compute the value of the right-hand side and to store that value in the storage location designated by the left-hand side.

The result will be stored in fixed or floating point form according as the variable on the left-hand side is a fixed or floating point variable.

If the variable on the left is fixed point and the expression on the right is floating point, the result will first be computed in floating point and then truncated and converted to a fixed point integer. Thus, if the result is ±3.569 the fixed point number stored will be ±3, not ±4.

Examples of Arithmetic Formulas

FORMULA	MEANING
A=B	Store the value of B in A.
I =B	Truncate B to an integer, convert to fixed point, and store in I.
A=I	Convert I to floating point and store in A.
I =I + 1	Add 1 to I and store in I. This example illustrates the point that an arithmetic formula is not an equation but a command to replace a value.
A=3.0*B	Replace A by 3B.

Control Statements

The second class of FORTRAN statements is the set of eight control statements, which enable the programmer to state the flow of his program.

Unconditional GO TO

GENERAL FORM	EXAMPLES
"GO TO n" where n is a statement number.	GO TO 3

This statement causes transfer of control to the statement with statement number n .

Computed GO TO

GENERAL FORM	EXAMPLES
"GO TO ($n_1, n_2, \ldots, n_m$), i" where $n_1, n_2, \ldots, n_m$ are statement numbers and i is a non-subscripted fixed point variable.	GO TO (30, 40, 50, 60), I

If at the time of execution the value of the variable i is j, then control is transferred to the statement with statement number n_j. Thus, in the example, if I has the value 3 at the time of execution, a transfer to statement 50 will occur.

This statement is used to obtain a computed many-way fork. A maximum of nine branches may be used in any one of these statements.

IF

GENERAL FORM	EXAMPLES
"IF (a) n_1, n_2, n_3" where a is any expression and n_1, n_2, n_3 are statement numbers.	IF (A(J, K)-B) 10, 20, 30

Control is transferred to the statement with statement number n_1, n_2, or n_3 according as the value of the expression a is less than, equal to, or greater than zero.

PAUSE

GENERAL FORM	EXAMPLES
"PAUSE" or "PAUSE n " where n is any unsigned fixed point constant	PAUSE PAUSE 1234

A PAUSE statement compiles as a stop command. During execution of the object program, the machine will halt. A subsequent depression of the program start key will cause resumption of operation at the point in the object program corresponding to the next FORTRAN statement. The n part of the PAUSE statement is disregarded by FOR TRANSIT but may be included for compatibility with 704 FORTRAN.

STOP

GENERAL FORM	EXAMPLES
" STOP " or " STOP n "	STOP STOP 1234

A STOP statement compiles as a stop command. During execution of the object program, the machine will halt. A subsequent depression of the program start key will cause the resumption of operation at the point in the object program corresponding to the next FORTRAN statement. The n part of the STOP statement is disregarded by FOR TRANSIT but may be included for compatibility with 704 FORTRAN.

NOTE: PAUSE and STOP are identical in the FOR TRANSIT system.

DO

GENERAL FORM	EXAMPLES
"DO n i = m_1, m_2" or "DO n i = m_1, m_2, m_3" where n is a statement number, i is a non-subscripted fixed point variable, and m_1, m_2, m_3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If m_3 is not stated it is taken to be 1.	DO 30 I = 1, 10 DO 30 I = 1, M, 3

The DO statement is a command to "DO the statements which follow, to and including the statement with statement number n, repeatedly, the first time with i = m_1 and with i increased by m_3 for each succeeding time; after they have been done with i equal to the highest of this sequence of values which does not exceed m_2 let control reach the statement following the statement with statement number n ."

The range of a DO is the set of statements which will be executed repeatedly; it is the sequence of consecutive statements immediately following the DO, to and including the statement numbered n .

The index of a DO is the fixed point variable i, which is controlled by the DO in such a way that its value begins at m_1 and is increased each time by m_3 until it is about to exceed m_2 . Throughout the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. During the last execution of the range, the DO is said to be satisfied.

Suppose, for example, that control has reached statement 10 of the program

```
10  DO 11 I = 1, 10
11  A (I) = I*N (I)
12
```

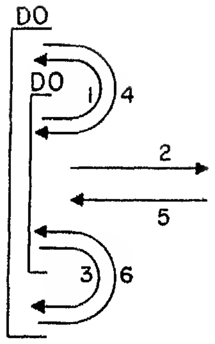
The range of the DO is statement 11, and the index is I. The DO sets I to 1 and control passes into the range. 1N(1) is computed, converted to floating point, and stored in A(1). Now, since statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2 and control returns to the beginning of the range, statement 11. 2N(2) is computed and stored in A(2). This continues until statement 11 has been executed with I = 10. Since the DO is satisfied, control now passes to statement 12.

DOs within DOs Among the statements in the range of a DO may be other DO statements. When this is so, the following rule must be observed:

Rule: If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former. A set of DOs satisfying this rule is called a nest of DOs. A nest must not exceed a depth of four DOs.

Transfer of Control and DOs Transfers of control by IF-type or GO TO-type statements are subject to the following rule:

Rule: No transfer is permitted into the range of any DO from outside its range. Thus, in the configuration below, 1, 2 and 3 are permitted transfers, but 4, 5 and 6 are not.



EXCEPTION: There is one situation in which control can be transferred into the range of a DO from outside its range. Suppose control is somewhere in the range of one or more DOs, and that it is transferred to a section of the program, completely outside the nest to which those DOs belong, which makes no change in any of the indexes or indexing parameters (m's) in the nest. Then after the execution of this section of program, control can be transferred back to the "same part of the nest" from which it originally came. (By "same part of the nest" is meant that no DO, and no statement which is a last statement in the range of a DO, shall lie between the exit point and re-entry point.) This provision makes it possible to exit temporarily from the range of a DO to execute a subroutine.

Preservation of Index Values The current values of all the indexes controlled by DO's are preserved for any subsequent use, until redefined.

For compatibility with 704 FORTRAN, after a normal exit from a DO, the value of the index controlled by that DO is not defined and cannot be used until redefined. A normal exit is defined as control passing to the statement following the range after the DO statement is satisfied.

Restriction on Calculations in the Range of a DO The only type of statement not permitted is one which redefines the value of the index or of any of the indexing parameters (m's); the indexing of a DO loop must be set before the range is entered. This indexing is accomplished by the DO statement. FOR TRANSIT II utilizes indexing registers which cannot be set by data cards, and the values of m_1, m_2, and m_3 must be < 2000.

CONTINUE

GENERAL FORM	EXAMPLES
"CONTINUE"	CONTINUE

CONTINUE is a dummy statement which gives rise to no instructions in the object program. A frequent use is as the last statement in the range of a DO to fill the requirement that the last statement in the range cannot be a transfer statement. As an example of a program which requires a CONTINUE, consider the table search program

```
10  DO 12I = 1, 100
11  IF (ARG-VALUE(I)) 12, 20, 12
12  CONTINUE
13
```

This program will examine the 100-entry VALUE table until it finds an entry which equals ARG, whereupon it will exit to statement 20 with the successful value of I available for fixed point use; if no entry in the table equals ARG a normal exit to statement 13 will occur. The following program

```
10  DO 11 I = 1, 100
11  IF (ARG-VALUE(I)) 11, 20, 11
12
```

would not work since DO-sequencing does not occur if the last statement in the range of a DO is a transfer.

END

GENERAL FORM	EXAMPLES
"END"	END

The END statement must be the last statement of the program. (The END statement for 704 FORTRAN requires additional information.)

Input-Output Statements

The FOR TRANSIT system presently allows for input and output of data by means of punched cards using the FORTRAN statements READ and PUNCH which are described below and in Chapter IV.

LIST: Three types of variables may be specified in a READ or PUNCH statement "LIST".

1. Non-subscripted variable

2. One member of an array; ELMNT (2, 5), or ELMNT (2, J), or ELMNT (I, J)

3. An entire array, by giving only the name of the array, ELMNT.

A maximum of ten variable names may be given in a "LIST".

READ

GENERAL FORM	EXAMPLES
"READ, LIST" or "READ n, LIST" where n may be a 1-4 digit fixed point constant and LIST is as described above.	READ, A, B, C READ 1, A, B, C READ 52, X, Y

The READ statement causes the object program to read cards. Record after record (card after card) is read until the complete List has been brought in and stored. The n portion of the READ statement is optional in the FOR TRANSIT system but may be included if compatibility with the FORTRAN system for the IBM 704 is desired.

PUNCH

GENERAL FORM	EXAMPLES
"PUNCH, LIST" or "PUNCH n , LIST" where n may be a 1-4 digit fixed point constant and LIST is as described above.	PUNCH, ROOT 1, ROOT 2 PUNCH 1, ROOT PUNCH 32, ARRAY PUNCH 1, ELMNT (2, 5)

The PUNCH statement causes the object program to punch cards. Record after record (card after card) is punched until the complete List has been punched. The n portion of the PUNCH statement is optional in the FOR TRANSIT system but may be included if compatibility with the 704 FORTRAN system is desired.

Conditional PUNCH

As a diagnostic convenience, a conditional form of the PUNCH statement is provided. Any PUNCH statement which is not numbered will produce, in the object program, a Punch command which can be controlled by the setting of the storage entry sign switch: In the object program with the sign switch set to minus (-), punching occurs; when set to plus (+), punching is bypassed.

Specification Statements

DIMENSION

GENERAL FORM	EXAMPLES
"DIMENSION v, v, v, . . ." where each v is a variable subscripted with 1 or 2 unsigned fixed point constants. Any number of v's may be given.	DIMENSION A(10), B(5, 15), C(3, 4)

The DIMENSION statement provides the information necessary to allocate storage in the object program for arrays of quantities.

Every variable which appears in the program in subscripted form must appear in a DIMENSION statement, and the DIMENSION statement must precede the first appearance of the variable. In the DIMENSION statement are given the desired dimensions of the array; in the executed program the subscripts of that variable must never take on values larger than those dimensions.

Thus the example states that B is a 2-dimensional array and that the subscripts of B will never exceed 5 and 15; it causes 75 words of storage to be set aside for the B array.

A single DIMENSION statement may be used to specify the dimensions of any number of arrays.

EQUIVALENCE

GENERAL FORM	EXAMPLES
EQUIVALENCE (A, B, C, . . .), (D, E, F, . . .) where A, B, C, D, ... are subscripted variables.	EQUIVALENCE (A, B, C), (D, E)

The EQUIVALENCE statement enables the programmer, if he wishes, to economize on data storage requirements by causing storage locations to be shared by two or more quantities when the logic of his program permits. It also permits him to call the same quantity by several different names, and then insure that those names are treated as equivalent.

The following rules must be observed when using the EQUIVALENCE statement with the FOR TRANSIT system.

1. EQUIVALENCE statements are restricted to subscripted variables.

2. EQUIVALENCE can only be specified which equates the first member of each of the respective arrays. Successive members will be automatically equated. If one of the arrays in an EQUIVALENCE statement has fewer members than the array it is being equated with, the EQUIVALENCE ends with the last member of the smaller array.

3. The EQUIVALENCE statement for a given set of variables must immediately follow the DIMENSION statement defining the set.

4. Any number of equivalences (pairs of parentheses) may be given in an EQUIVALENCE statement.

5. Each equivalence set is limited to five subscripted variable names, e.g., (A, B, C, D, E) is a maximum equivalence set.

6. A variable name can appear only once in an EQUIVALENCE statement.

The sharing of storage locations cannot be planned safely without a knowledge of the two types of FORTRAN statements which when executed at object program time will cause a new value to be stored in a storage location.

1. Execution of an arithmetic statement stores a new value of the variable on its left-hand side.

2. Input statements store new values of the variables listed.

Summary of FORTRAN Sequencing

The precise laws which govern the order in which the statements of a FORTRAN program will be executed, and which have been left unstated up to this point, may be stated as follows:

1. Control begins at the first executable statement.

2. If control is in a statement S, then control will next go to the statement dictated by the normal sequencing properties of S.

3. EXCEPTION. If, however, S is the last statement in the range of one or more DOs which are not yet satisfied, and if S is not a transfer (IF-type or GO TO-type statement), then the normal sequencing of S is ignored and DO-sequencing occurs, i.e., control will next go to the first statement of the range of the nearest of the unsatisfied DOs, and the index of that DO will be raised.

4. The statements DIMENSION and EQUIVALENCE are non-executable statements, and in any question of sequencing are simply to be ignored.

5. The last statement of each source program must be an END statement.

Examples and Explanatory Notes

This section, which includes some brief and relatively simple examples of FORTRAN programs with pertinent comments, is provided to illustrate the process of writing FORTRAN statements and using them to form a meaningful program. One of the sample problems shown here appears in Appendix B where it is carried through each phase of processing to provide a sample of the input and output for each step, including the running of the object program.

1. It may be helpful to refer first to the formula for one of the roots of a quadratic equation which was mentioned briefly on the first page of this chapter. The algebraic representation for a specific case might be written

$$\text{root} = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

where
$A = +3$
$B = +1.7$
$C = -31.92$

A complete FORTRAN program to provide for making this calculation and punching the result may be written in six separate statements as follows:

C ← FOR COMMENT / STATEMENT NUMBER 1 5	CONTINUATION 6	7 FORTRAN STATEMENT 72
		A = 3.
		B = 1.7
		C = -31.92
		ROOT = (-B +SQRTF(B**2-4.*A*C))/(2.*A)
11		PUNCH 1, ROOT
		END

The first statement means "assign the value 3. to the variable A." The next two statements have a similar meaning. The fourth statement means "evaluate the expression on the right side and assign the result to the variable ROOT." The last statement instructs the computer to stop.

The sequential nature of the program should be noted. The computer executes instructions in the same order as the order of the statements. For example, if the fourth statement were moved up and made the first statement, then the computer would evaluate ROOT before obtaining the desired values of A, B, and C. ROOT would, therefore, be evaluated using some arbitrary, unknown values for these variables. The same result could be obtained of course by writing

C ← FOR COMMENT / STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72
		ROOT = (-1.7 +SQRTF(1.7**2-4.*3.*(-31.92)))/(2.*3.)
11		PUNCH 1, ROOT
		END

in which the actual numerical values appear in the statement describing the evaluation of ROOT.

Obviously, the foregoing is applicable only to a specific case. A more likely form of the program is one in which it is possible to process many sets of data and obtain a root for each set of values of A, B, and C.

Such a program might be written

C ← FOR COMMENT / STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72
11		READ 1, A, B, C
		ROOT = (-B +SQRTF(B**2-4.*A*C))/(2.*A)
12		PUNCH 1, ROOT
		GO TO 11
		END

The first statement causes the computer to read a data card in which the values of A, B, and C have been punched. The next two statements remain unchanged from the first program. The fourth statement provides a transfer to the READ statement, and thus causes the entire process to be repeated. The last statement is an END statement which is required in a FOR TRANSIT program. The object program resulting from this FORTRAN program will read data cards, compute the roots, and punch out answer cards as long as there are data cards remaining in the read hopper of the 533 Read-Punch Unit.

2. In this example, a program is required to determine the current in an alternating current circuit consisting of a resistance, an inductance, and a capacitance, having been given a number of sets of values of resistance, inductance, and frequency. The

current is to be determined for a number of equally spaced values of the capacitance (which lie between specified limits) for voltages of 1.0, 1.5, 2.0, 2.5, and 3.0 volts.

The equation for determining the current flowing through such a circuit is

$$i = \frac{E}{\sqrt{R^2 + \left[2\pi fL - \frac{1}{2\pi fC}\right]^2}}$$

where	i = current, amperes
	E = voltage, volts
	R = resistance, ohms
	L = inductance, henrys
	C = capacitance, farads
	f = frequency, cycles per second
	π = 3.1416

The FORTRAN program could be written as follows:

```
C ← FOR COMMENT
STATEMENT   CONTINUATION   FORTRAN STATEMENT
NUMBER
1        5  6              7                                              72
   10                      READ 1, OHM, FREQ, HENRY
   11                      READ 1, FRD 1, FRDMX
   12                      VOLT = 1.0
   15                      FARAD = FRD 1
   14                      PUNCH 1, VOLT
   16                      AMP = VOLT/SQRTF(OHM**2 +(6.2832*FREQ*HENRY
            1                -1./(6.2832*FREQ*FARAD))**2)
   17                      PUNCH 1, FARAD, AMP
   18                      IF (FARAD - FRDMX) 19, 21, 21
   19                      FARAD = FARAD + .00000001
   20                      GO TO 16
   21                      IF (VOLT - 3.0) 22, 10, 10
   22                      VOLT = VOLT +0.5
   23                      GO TO 15
   24                      END
```

Statement 10 causes the values of the resistance, the frequency, and the inductance to be read from the first card, and statement 11 causes the initial and final values of the capacitance to be read from the next card. The initial value of the voltage is introduced and punched (statements 12 and 14). Statement 15 causes the initial value of the capacitance to replace the "current" value of the capacitance (denoted as FARAD). The actual calculation is specified by statement 16. The result of that calculation, together with the current value of the capacitance, is then punched (statement 17).

The current value of the capacitance is compared with the final value to determine whether or not all values have been investigated (statement 18). If not, the expression is negative and the program proceeds to statement 19, which causes the value of the capacitance to be increased by the given increment. This is followed by a transfer (statement 20) to statement 16 which causes the calculation to be repeated for the new value of the capacitance. If the expression in statement 18 is zero or positive, all values of the capacitance have been investigated and the program transfers to statement 21.

At this point the value of the voltage is compared with the upper bound to determine whether or not all specified values of the voltage have been used. If not, the expression in statement 21 is negative and the program proceeds to statement 22 which causes the value of the voltage to be increased. Following this, a transfer (statement 23) is made to statement 15, causing the new value of the voltage to be punched; and the entire process of investigating all values of the capacitance is begun again. If all values of the voltage have been used (the expression in statement 21 is zero or positive), the calculations for the current set of values of resistance, frequency, and inductance are finished. The program is returned to statement 10 so that the two cards defining the next case may be read and the program repeated. This process is repeated until all of the cases have been considered, i.e., all of the cards have been read.

3. The problem in this example may be stated as follows:

Given X_i, Y_i, Z_j for $i = 1, 10$ and $J = 1, 20$ to compute:

$$PROD = \left(\sum_{i=1}^{10} A_i\right)\left(\sum_{j=1}^{20} Z_j\right)$$

where
$$\begin{aligned} A_i &= X_i^2 + Y_i & &\text{if } |X_i| > |Y_i| \\ A_i &= X_i + Y_i^2 & &\text{if } |X_i| < |Y_i| \\ A_i &= 0 & &\text{if } |X_i| = |Y_i| \end{aligned}$$

A possible FORTRAN program follows:

C ← FOR COMMENT STATEMENT NUMBER 1 5	CONTINUATION 6	FORTRAN STATEMENT 7 72
3		DIMENSION X(10), Y(10), Z(20)
5		READ 1, X, Y, Z
6		SUM A = 0.0
7		DO 12 I = 1, 10
8		IF (ABSF(X(I))- ABSF(Y(I))) 9, 12, 11
9		SUM A = SUM A + X(I) + Y(I)**2
10		GO TO 12
11		SUM A = SUM A + X(I)**2 + Y(I)
12		CONTINUE
13		SUM Z = 0.0
14		DO 15 J = 1, 20
15		SUM Z = SUM Z + Z(J)
16		PROD = SUM A * SUM Z
17		PUNCH 1, SUM A, SUM Z, PROD
18		END

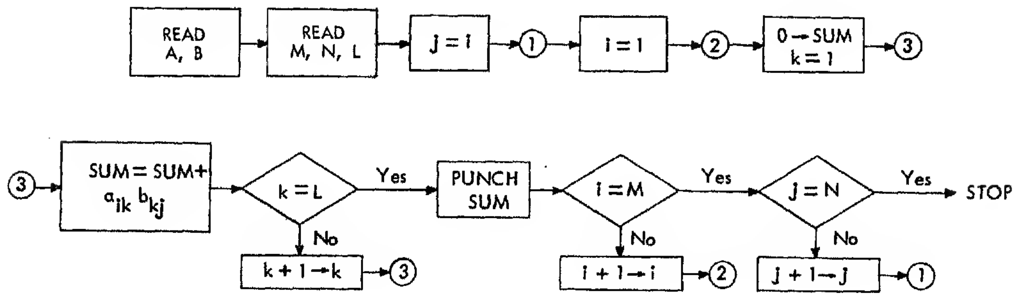
The DIMENSION statement sets aside storage locations for the input data. The READ statement reads the input data from cards into the 650. Statement 6 sets the quantity SUM A to zero. Statements 8-12, under control of the DO (statement 7), compute $\sum_{i=1}^{10} A_i$. Statement 15 computes $\sum_{j=1}^{20} Z_j$ under the control of DO statement 14. The following statements compute and punch PROD. Statement 12, CONTINUE, serves as a common reference point; and since it is the last statement in the range of the DO, after its completion I is increased and the next repetition begun.

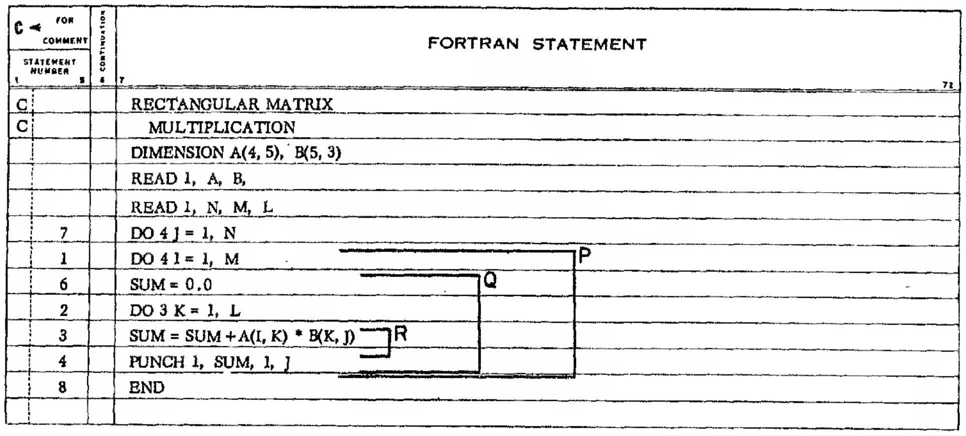
4. In this last example a program is required to perform the following operations: Multiply the L x N matrix b_{kj} by the M x L matrix a_{ik} obtaining the product elements by the relation

$$c_{ij} = \sum_{k=1}^{L} a_{ik} \cdot b_{kj}$$

Punch the elements as they are computed.

The problem may be represented in flow chart form as follows:





C ◄ FOR COMMENT / STATEMENT NUMBER 1–5		CONTINUATION 6	FORTRAN STATEMENT 7–72
C			RECTANGULAR MATRIX
C			MULTIPLICATION
			DIMENSION A(4, 5), B(5, 3)
			READ 1, A, B,
			READ 1, N, M, L
	7		DO 4 J = 1, N
	1		DO 4 I = 1, M
	6		SUM = 0.0
	2		DO 3 K = 1, L
	3		SUM = SUM + A(I, K) * B(K, J)
	4		PUNCH 1, SUM, I, J
	8		END

The dimension statement reserves storage for the two matrices, A and B, which in this example are 4 X 5 and 5 X 3, respectively. For explanatory purposes only, three brackets have been drawn around parts of the program to show the sequence of statements controlled by each DO statement. The brackets P, Q, and R correspond to the connectors①, ②, and③ in the flow chart. The first DO statement specifies that procedure P, i.e., the following statements down to statement 4, is to be carried out for J = 1, then J = 2 and so on until J = N. The first statement of procedure P (DO 4 I = 1, M) directs that procedure Q be done for I = 1 to I = M. And of course each execution of procedure Q involves L executions of procedure R for K = 1, 2, . . ., L.

Consider procedure Q. Each time its last statement is completed, the element c_{ij} (called SUM) has been punched, the "index" I of its controlling DO statement is increased by 1 and control goes to the first statement of Q until finally its last statement is reached with I = M. Since this is also the last statement of P and P has not been completed until J = N, J will be increased by 1 and control will then pass to the first statement of P. This statement (DO 4 I = 1, M) causes the repetition of Q to begin again. Finally, the last statement of Q and P (statement 4) will be reached with I = M and J = N, meaning both Q and P have been repeated the required number of times. Control will then go to the next statement, "END".

Each time R is executed a new term is added to a product element. Each time Q is executed a new product element is punched. Each time P is executed a product column has been completed.

CHAPTER II - INCORPORATING SUBROUTINES

The ability to evaluate a function is not built into the IBM 650; there is no operation code equivalent to $\sqrt{\ }$. If such a function be included in a FOR TRANSIT statement a program must be available to evaluate it. This program is referred to as a subroutine. Up to ten such subroutines may be used in any one FOR TRANSIT program. This chapter provides the necessary instructions and relevant information for adapting subroutines and including them in the system.

Built-in Subroutines

The FOR TRANSIT system has built into it a number of subroutines for the most commonly used functions. Subroutines for floating fixed point numbers and fixing floating point numbers, for handling the various arithmetic operations, and for performing the input-output operations are included in the Package Deck. These subroutines in the form of machine language instructions are loaded into the 650 at the time of running the object program. A complete list of the built-in subroutines will be found at the end of this chapter. Although designed for the specific purposes indicated, the built-in subroutines may be utilized by the programmer by preparing an appropriate function title card.

Adding Function Subroutines

Subroutines added to the system by the user for the purpose of evaluating functions will be integrated into the object program as follows:

1. The translator translates the function name found in the FORTRAN program into corresponding IT notation. The IT notation for built-in subroutines is supplied by the system.

2. The compiler generates the symbolic entry to the appropriate subroutine using the IT notation supplied by the translator.

3. The assembler reserves sufficient storage space for the subroutines.

The programmer must (1) specify the names of the functions being used, (2) specify the IT call names which he wishes to have associated with the function names and which will determine the symbolic entries to the subroutines, and (3) include the subroutines themselves, either in symbolic, i.e., SOAP II, form or in absolute form. These requirements are considered individually in subsequent paragraphs.

1. Each function name which appears in a FOR TRANSIT source program (see page 11 for general form and examples) must be defined by a function title card when the source program is processed. The function title card contains the FOR TRANSIT language name of the function as it appears in the source program, e.g., SQRTF, and the IT call name to be associated with it in the compiling phase, e.g., 23EK. A description of the card format for function title cards is included in Chapter III. The function title cards are loaded by the translator prior to the statement cards.

2. The IT call name which is punched in its corresponding function title card is of the form nEK, where n is a one, two, or three-digit number. The choice of subroutine number n is arbitrary except that (a) none of the numbers of the built-in subroutines may be used, (b) no number may be used twice, and (c) n must be less than 500 if the output is in floating point form and equal to or greater than 500 if the output is in fixed point form.

3. Function subroutines may be included in the system in either of the two forms indicated below.

   a. A function subroutine in SOAP II symbolic form is always incorporated into the object program by assembling the symbolic instructions of the subroutine together with the symbolic instructions produced by the compilation phase.

      Symbolic addresses used in the subroutines must conform to the following rules:

      1. The entry point must correspond to the entry generated by the system. The entry generated from the IT call name (nEK) will be of the form E00ab where ab is a pair of alphabetic characters derived from the subroutine number by the following method: The subroutine number is divided by 26; both the resulting quotient and remainder are incremented by one, and then each is transposed into its alphabetic equivalent (1 = A, 2 = B, etc.) to form the pair. Thus subroutine number 5 translates into the pair AF, the number 80 into the pair DC, and the number 571 into the pair VZ.

      2. The entry point must be the first card of the subroutine at assembly time.

3. The first character must not be the letter "L".

4. Symbolic addresses used by more than one subroutine must begin with the letter "E".

b. A function subroutine in absolute form is in the standard five instructions per card format, and is incorporated into the object program at the assembly phase. A SYN card in SOAP II format is required for the entry point of the subroutine. For example, if the absolute location of the entry point is 0500 and the location as generated by the system is E00AB the SYN card is as follows:

COL	48- -50	51- -55	56	57	58- -61	80
	SYN	E00AB			0500	

Writing Function Subroutines

The subroutines to be incorporated into a FOR TRANSIT program must be written to conform to the format required by the system. The system assumes the following entry and exit conditions for all subroutines:

Let the subroutine be a function of "k" variables in the order: V_1, V_2, . . . , V_k; where V_n is an expression, variable, or constant. (n = 1, 2, . . . , k)

a. The entry conditions are as follows:

1. V_1 is in the lower accumulator and the upper accumulator is zero for FOR TRANSIT I and I(S), and for the fixed mode of FOR TRANSIT II and II(S).

2. V_1 is in the upper accumulator and the lower accumulator is zero for the floating mode of FOR TRANSIT II and II(S).

3. V_n is stored in P0000 + (k - n) when $k > 1$.

4. The contents of the distributor are:

k = 1	Exit instruction
k > 1	00 P0000 + (k - 1) 0000 + m

Contents of "m" is the next instruction.
Contents of P0000 + (k - 1) is V_2.

5. If V_n is itself a function of variables:

   W_1, W_2, . . . , W_r; then

   a. W_1 is in the lower or upper accumulator as per rules for V_1 in 1. and 2. above.

   b. W_r is stored in P0000 +(k - n)

   c. W_j is stored in P0000 +(k - n + r - j)

   d. At entry the D address of the distributor is the location of W_2, and the I address is the location of the next instruction.

   e. The result of V_n is stored in P0000 +(k - n).

b. The exit conditions are as follows:

   1. The result is in the lower accumulator and the upper accumulator is zero for FOR TRANSIT I and I(S) and when the mode of the subroutine is fixed for FOR TRANSIT II and II(S).

   2. The result is in the upper accumulator and the lower accumulator is zero when the mode of the subroutine is floating for FOR TRANSIT II and II(S).

The maximum number of arguments, V_n , is thirteen. These arguments are evaluated from right to left. Each V_n can itself be a function of variables, W_j . The argument V_n to the extreme right can be a function of thirteen variables as there are 13 locations available at the time the first argument is evaluated. The argument to its left, however, can be a function of not more than twelve variables as one of the thirteen available locations is now taken by the argument that has been evaluated. In short, the subroutine may have arguments V_1, V_2, V_3, . . . , V_k where $k \leq 13$, and each V_n can be a function of r variables where $r \leq n$ if $k = 13$, $r \leq (n + 13 - k)$ if $k < 13$.

A listing of the program for the following statement is included in Appendix B to illustrate how the parameter, entry and exit are handled by the system:

Y = OUTF (A, B, C, D, ENRF (E, F, G), H)

Use of Indexing Registers : FOR TRANSIT II and II (S)

The built-in subroutines include a routine to store and restore the indexing registers. This routine may be used by incorporated routines as follows:

1. Store the EXIT instruction of the incorporated routine in "ERTHX".

2. Load distributor with the next instruction of the incorporated routine and transfer control to "EZZZA". The contents of the indexing registers will be stored and the instruction in the distributor executed.

3. After executing the incorporated routine, transfer control to EZZZB. The indexing registers will be restored and the EXIT instruction in ERTHX executed.

The object program in FOR TRANSIT II and II(S) utilizes the indexing registers when under the control of a DO statement. Incorporated subroutines which use indexing registers and occur within the range of a DO statement must preserve the current values of the indexing registers.

Example of Incorporated Subroutine

To incorporate the sample subroutine with the FORTRAN call name "DCAF" and the IT compiler "28EK":

TRANSLATION PHASE: Prepare a Function Title Card (see page 40). This card is loaded preceding the FORTRAN statements.

COMPILATION PHASE: Nothing required.

ASSEMBLY PHASE: The subroutine may be incorporated in symbolic SOAP II format or absolute five instruction per card format.

1. To include the subroutine in symbolic SOAP II format the entry point of the routine which is the first card of the routine will be E00BC.

2. To include the subroutine in absolute five instruction per card format a SYN card is required for the entry point, (SYN E00BC nnnn).

Other Requirements

Subroutines incorporated into the FOR TRANSIT I(S) or II(S) systems in symbolic SOAP II format require a "12" punch in card column 5 for correct read-in of the cards. Card columns 7-36 and 73-75 must be blank.

Absolute subroutines incorporated into a FOR TRANSIT program are read in as load cards and require a "12" punch in card column 2.

List of Subroutines in Package Deck for FOR TRANSIT I

IT No.	Purpose of Subroutine	Symbolic Entry
4	(L) Floating Point $\leftarrow$ (L) Fixed Point	E00AE
5	(L) and (ACC) Floating Point $\leftarrow$ (L) Fixed Point	E00AF
6	(L) and (ACC) $\leftarrow$ (L) / (ACC)	E00AG
8	(L) and (ACC) $\leftarrow$ (L) + (ACC)	E00AI
9	(L) and (ACC) $\leftarrow$ (L) x (ACC)	E00AJ
14	(L) and (ACC) $\leftarrow$ (ACC) / (L)	E00AO
16	READ	E00AQ
17	PUNCH	E00AR
501	(L) Fixed Point $\leftarrow$ (L) Floating Point	E00TH
10	(L) and (ACC) $\leftarrow$ (L) Fixed$^{\text{(ACC) Fixed}}$	E00AK
11	(L) and (ACC) $\leftarrow$ (L) Floating$^{\text{(ACC) Fixed}}$	E00AL
302	(L) and (ACC) $\leftarrow$ (L) Floating$^{\text{(ACC) Floating}}$	E00LQ
1	(L) $\leftarrow$ Log_{10} (L)	E00AB
2	(L) $\leftarrow$ $10^{(L)}$	E00AC
300	(L) $\leftarrow$ Log_e (L)	E00LO
301	(L) $\leftarrow$ $e^{(L)}$	E00LP

NOTE: In the above listing, the second and third characters of the symbolic entries are zeros, not alphabetic O's. The notation (L) refers to the contents of the lower accumulator, and (ACC) to the contents of the pseudo floating point accumulator which occupies drum storage location 0000.

List of Subroutines in Package Deck for FOR TRANSIT II

IT No.	Purpose of Subroutine	Symbolic Entry
0	Floating Arithmetic Device Overflow-Underflow checking and correcting routine (see note 2 below).	E00AA
4	(U) Floating Point $\leftarrow$ (L) Fixed Point	E00AE
5	(U) and (ACC) Floating Point $\leftarrow$ (L) Fixed Point	E00AF
16	READ	E00AQ
17	PUNCH	E00AR
501	(L) Fixed Point $\leftarrow$ (U) Floating Point	E00TH
10	(L) and (ACC) $\leftarrow$ (L) $\text{Fixed}^{\text{(ACC) Fixed}}$	E00AK
11	(U) and (ACC) $\leftarrow$ (U) $\text{Floating}^{\text{(ACC) Fixed}}$	E00AL
302	(U) and (ACC) $\leftarrow$ (U) $\text{Floating}^{\text{(ACC) Floating}}$	E00LQ
1	$(U) \leftarrow \text{Log}_{10}(U)$	E00AB
2	$(U) \leftarrow 10^{(U)}$	E00AC
300	$(U) \leftarrow \text{Log}_e\ (U)$	E00LO
301	$(U) \leftarrow e^{(U)}$	E00LP

NOTE: 1. In the above listing, the second and third characters of the symbolic entries are zeros, not alphabetic O's. The notation (L) refers to the contents of the lower accumulator, (U) to the contents of the upper accumulator, and (ACC) to the contents of the pseudo floating point accumulator which occupies drum storage location 0000.

2. The compilation phase will generate an entry to Subroutine E00AA at the replacement symbol (=) of each Arithmetic Statement, except when the Storage Entry Sign Switch is set to Minus (-). Subroutine E00AA will stop the program if underflow or overflow has occurred. Depressing Program Start will cause the program to continue.

CHAPTER III - PROCESSING THE SOURCE PROGRAM

This chapter includes the necessary information and instructions for processing a FOR TRANSIT program to obtain the object, or machine language, program. The first sections of the chapter deal with the preparation of statement cards and function title cards, and subsequent sections constitute operator's instructions and notes for each of the three phases of the FOR TRANSIT system.

Source programs stated in the FORTRAN language described in the preceding chapter may be written on standard FORTRAN coding sheets, IBM Form X28-7327. These are stocked at IBM Stationery Stores in Endicott, New York, and may be ordered through local sales representatives. The use of the coding forms is encouraged to avoid programming errors and to facilitate the transcription of the FORTRAN statements to punched card form as described in the following section.

Preparing the Statement Cards

FOR TRANSIT statements are punched in modified FORTRAN statement cards using either a 24 or a 26 Card Punch Machine. Regular FORTRAN cards, IBM electro number 888157, may be used. Two different card formats are provided; the choice between formats is dictated by the equipment specifications of the 650 to be used for processing, as described in subsequent paragraphs. Regardless of which format is used, each FORTRAN statement is punched on a separate card using the standard FORTRAN code shown below.

STANDARD FORTRAN CODE

Char	Punch	650	Char	Punch	650	Char	Punch	650	Char	Punch	650
1	1	91	A	12-1	61	J	11-1	71	/	0-1	31
2	2	92	B	12-2	62	K	11-2	72	S	0-2	82
3	3	93	C	12-3	63	L	11-3	73	T	0-3	83
4	4	94	D	12-4	64	M	11-4	74	U	0-4	84
5	5	95	E	12-5	65	N	11-5	75	V	0-5	85
6	6	96	F	12-6	66	O	11-6	76	W	0-6	86
7	7	97	G	12-7	67	P	11-7	77	X	0-7	87
8	8	98	H	12-8	68	Q	11-8	78	Y	0-8	88
9	9	99	I	12-9	69	R	11-9	79	Z	0-9	89
Blank		00	+	12	20	-	11	30	0	0	90
=	8-3	48	.	12-3-8	18	-	4-8	49	,	0-3-8	38
			)	12-4-8	19	*	11-4-8	29	(	0-4-8	39

NOTE: On the 24 and 26 Card Punch Machines equipped for special character punching, the character ⌑ is the equivalent of the character) ; % is the equivalent of (; & is the equivalent of +; and # is the equivalent of = . If desired, the 24 and 26 machines may be modified on an RPQ basis (Request Price Quotation) to include the "FORTRAN key tops and printing code plate." This includes @ equivalent to "—".

If a statement is too long to fit in the statement field of a single card, it may be continued over as many as nine additional (continuation) cards. The maximum statement length is 125 characters, exclusive of blank spaces. As blanks in the statement field are ignored by the translator, the programmer may use them freely to improve the readability of the source program listing. In any continuation cards, column 6 must not contain a zero or be left blank; it is used to number the continuation cards from 1 to 9. Other details of the card formats are as follows:

Card Format for FOR TRANSIT I and II

FOR TRANSIT I and II, both of which run on the basic 650 without a special character device, require the statement card format shown in Figure 2. The statement itself is punched in a twenty column field, columns 7 to 26. Numerical, alphabetic, and special characters and blanks may all be included in the FORTRAN statement. The FOR TRANSIT system will read and accept all of the punches without the aid of a special character device.

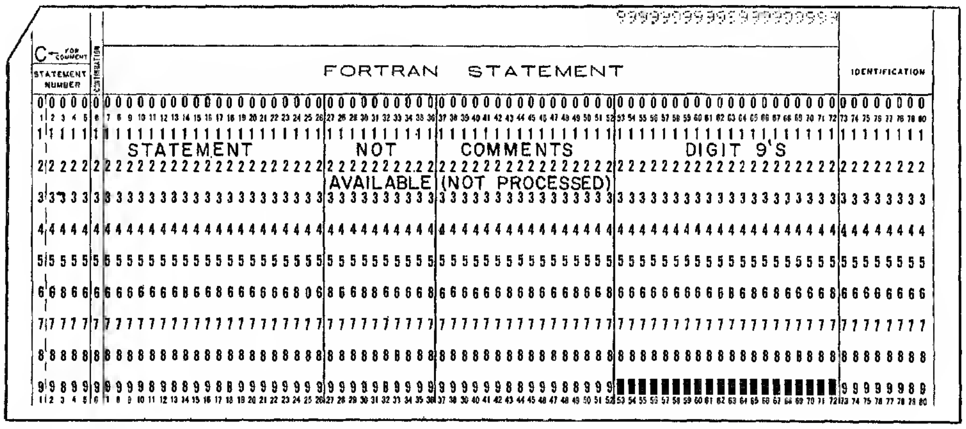


Figure 2

Card columns	Description
1	An alphabetic C (or any non-zero punch) in this column indicates a comments card, which will be ignored during translation. A zero indicates a statement to be translated. This column must have a punch in it.
2 - 5	Statement number. This field must contain numerical punches.

Card columns	Description
6	Used to indicate continuation cards. A zero indicates first card of a statement regardless of whether the statement uses one or more cards. A non-zero punch from 1-9 indicates a continuation card. This field must contain a numerical punch.
7-26	The statement. Numerical, alphabetic and special characters and blank columns are all acceptable in this field.
27-36	Not available (blanks).
37-52	Not read by FOR TRANSIT. May be used for comments.
53-72	A digit 9 must be punched in each of the 20 columns.
73-80	Identification. Not read by FOR TRANSIT.

<u>Card Format for FOR TRANSIT I (S) and II (S)</u>

Those installations in which the 650 is equipped with a special character device will utilize FOR TRANSIT I (S) or II (S), and the statement card format applicable to these systems is shown in Figure 3. It will be noted that this format is quite similar to the preceding format except that the statement field includes ten additional card columns and columns 37 through 80 must be blank.

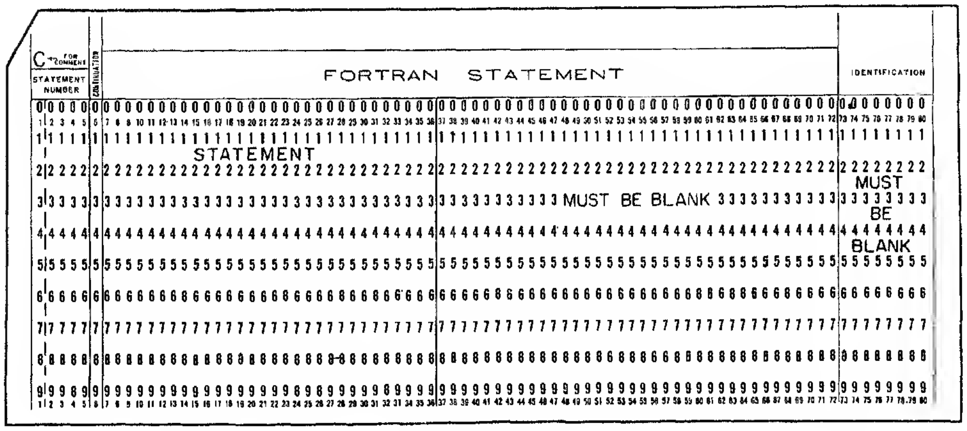


Figure 3

Card columns	Description
1	An alphabetic C in this column indicates a comments card, which will be ignored during translation. This field may be left blank.
2-5	Statement number. This field may be left blank.
6	Used to indicate continuation cards. Zero or blank indicates first card of a statement. A non-zero punch indicates a continuation card. This field may be left blank.
7-36	The statement. Numerical, alphabetic and special characters and blank columns are all acceptable in this field.
37-72	This field MUST BE LEFT BLANK.
73-80	MUST BE BLANK. Normally used for identification but reserved for zone punches.

Sample Problem Statement Cards

A machine listing of the statement cards for the matrix multiplication example discussed in Chapter I is included in Appendix B.

Preparing the Function Title Cards

Title cards for function subroutines are required in the translating phase to create an internal table of function names as described in the preceding chapter. The format of these title cards is shown in Figure 4.

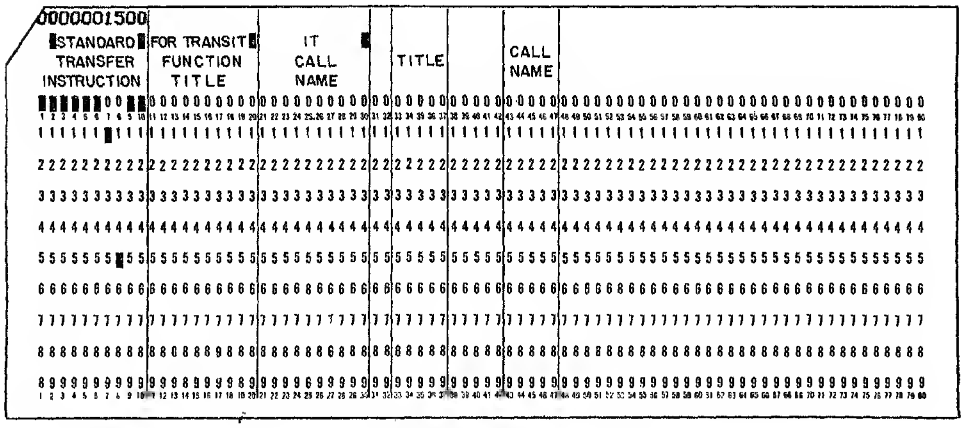


Figure 4

Card columns	Description
1-10	A standard instruction, 00 0000 1500, which is required in all function title cards. Columns 2 and 10 must each contain a 12-punch.
11-20	The FOR TRANSIT function name (title) in 650 double digit representation. A name consisting of less than five characters (ten digits) must be punched in the low order positions of the field. Unused high order positions of the field must be punched 00. A 12-punch is required in column 20.
21-30	The IT call name in 650 double digit representation. If a name consists of less than five characters (ten digits) by virtue of its having a subroutine number < 100, it must be left-justified in the field. Unused low order positions must be punched 00. A 12-punch is required in column 30.
31-32	Blank columns.
33-37	The FOR TRANSIT function title in single character representation. This field is for identification purposes only.
38-42	Blank columns.
43-47	The IT call name in single character representation. This field is for identification purposes only.
48-80	Blank columns.

A completed function title card is shown in Figure 5. In this sample card the function specified is the square root (SQRTF), and the subroutine to compute the function has been assigned the number 23. This card will cause the entry E00AX to be generated by the compiler; the appropriate subroutine must be incorporated at the SOAP level with the symbolic output of the compiler (see Chapter II).

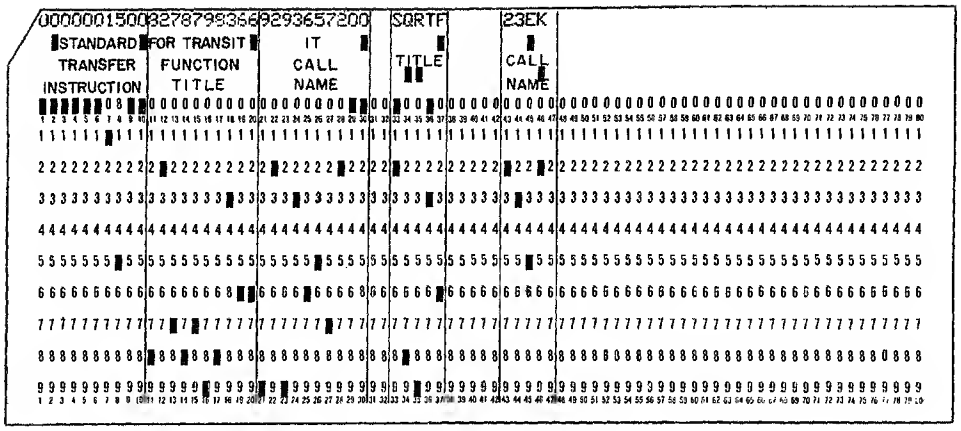


Figure 5

NOTE: Function title cards must be in numerical order with respect to the information punched in columns 11-20 of the cards when they are loaded in the translation phase; i.e., they must be in alphabetic order.

Operating Instructions Phase I

Phase I (Translation)

Console Settings

Storage Entry: 70 1952 9999 (If FOR TRANSIT and function title cards are already loaded, 00 0000 1999)

Switches:	Programmed	STOP
	Half Cycle	RUN
	Control	RUN
	Display	UPPER
	Overflow	SENSE
	Error	STOP

Operation

1. Ready machine with proper console settings, FOR TRANSIT 533 control panel and blank cards in the punch hopper.

2. Ready read hopper with:

   a. FOR TRANSIT deck
   b. Title cards (in numerical order on cols. 11-20) for any function subroutines required by the program
   c. FORTRAN statement cards

3. Depress computer reset key; program start key; and, when read hopper empties, end-of-file key.

The 650 will load the FOR TRANSIT deck and any function title cards and will automatically start reading the FORTRAN statement cards. Translation will take place on a statement by statement basis, with the corresponding IT statements being punched concurrently. The last FORTRAN statement to be translated should be a "END". Immediately after this has been processed the machine will punch a Header Card needed for the IT compilation step.

4. At the conclusion of the translation process, run cards out of the punch feed and discard the first and last cards. The remaining cards, in order, are (a) the IT statements, and (b) the IT Header Card.

5. Rearrange the card order to (a) the IT Header Card, (b) the IT statements. This is now the complete IT program and is ready for compiling.

Translating More Than One Program at a Time

It is not necessary to reload the FOR TRANSIT deck to translate additional source programs. Simply stack the statement card decks for the several source programs one after the other in the read feed as if they were all one program. The last statement of each program, i.e., the concluding "END" statement will cause the 650 to:

a. Punch the Header Card
b. Initialize and proceed with the translation of the next program in the read hopper.

This procedure assumes, of course, that appropriate function title cards are included (immediately after the FOR TRANSIT deck) to create an internal table of function titles containing all the functions which will be encountered in the several source programs.

To create a new table of function titles requires the loading of the FOR TRANSIT deck followed by the new function title cards.

Programmed Stops

The machine is programmed to stop under certain conditions during the translation phase. If a stop is encountered, display the contents of the program register and compare the data address with the following list to determine the reason for the stop.

Data Address	Reason for Stop
0001	Unacceptable or misspelled non-arithmetic statement.
0002	Table overflow, i.e., the limit on the number of variables (either subscripted or non-subscripted), or on the number of branches in computed GO TO statements has been exceeded.
0003	Statement with more than 125 characters.
0004	A variable contains more than 5 characters.
0005	The scan routine, which inserts additional pairs of parentheses to satisfy the requirements of the compiler, is attempting to exceed the limit of 15 pairs. (Though not a programmer's error, the statement must be rewritten as two statements and reprocessed.)
0006	More than ten function title cards.
0007	Function title cards not in ascending order.
0010	The statement does not contain an even number of parentheses.
0020	An unacceptable combination of punches in a card column.
0030	An unacceptable function name, i.e., one not defined by a function title card.
0040	The variable has been omitted from a control statement.
0100	More than 5 characters in the constant or variable component of a subscript.
0300	Translation results in a statement of more than 120 characters in length. (This is not a programmer's error, but is caused by the expansion from FORTRAN to IT. It nevertheless must be corrected by breaking up the offending statement into two statements in order to avoid compilation error.)

Data Address	Reason for Stop
0400	Improper DIMENSION statement.

Error Procedure

When an error is indicated, remove the cards from the read hopper and stacker and run out the cards still in the read unit. The first card out (or the fifth card from the back) will be the one containing the error. If the error card is a continuation card, the error is in the corresponding statement but not necessarily in that portion of the statement contained on this card. If corrections can be made at once, do so and reload the read hopper with the corrected card and all of the cards which follow it in the program and press the program start key. Translation will be resumed. If it is not possible to correct the error immediately, relinquish the 650 and completely reprocess at a later time. It is important to remember that an error indication appears at the time that the error is being processed for translation. Under certain circumstances, therefore, part of the statement containing the error may have been translated and punched out before the error was encountered (for example, in a lengthy arithmetic statement or in a PUNCH statement containing a series of variables). In such a case it is necessary to remove the erroneously punched cards from the punch stacker before restarting the program.

Operating Instructions Phase II

Phase II (Compilation)

As mentioned previously, the IT compiler has been modified for the purposes of the FOR TRANSIT system. Standard IT decks should not be used in conjunction with the FOR TRANSIT system.

Console Settings

Storage Entry:	70 1952 9999 (If IT is already loaded, 00 0000 1999)
Switches:	Same as for Phase I

Operation

1. Ready machine with proper console settings, FOR TRANSIT 533 control panel and blank cards in the punch hopper.

2. Ready read hopper with:

   a. IT deck
   b. IT Header Card
   c. the IT statements

3. Depress computer reset key; program start key; and, when read hopper empties, end-of-file key.

   The 650 will load the IT deck and begin reading the IT statements. Punching will occur as each statement is compiled and ready for output.

4. At conclusion of compilation, run the cards out of the punch feed, removing the first and last cards. The remaining cards are one reservation card for data requirements and the compiled symbolic program. The program is in one-instruction-per-card SOAP II format, with the IT statements in the remarks field. The deck is now ready for assembly.

Compiling More than One Program at a Time

It is not necessary to reload the IT deck to compile additional programs. Simply stack the statement card decks for the several problems, preceded by their respective Header Card, one after the other in the read feed as if they were all one program. The last statement of each program, i.e., the translated "END" statement, will cause the 650 to:

a. punch the problem constant table

b. initialize and proceed with the compilation of the next program in the stack.

Programmed Stops

The address lights will show 1234 for error stops. The four high order positions of the display lights (Upper) will specify the statement number of the offending statement; the next position will be zero; and the five low order positions will indicate the type of error according to the following code.

Code	Reason for Stop
01 0 01	The number of instructions compiled from a single statement exceeds 93 instructions.

Code	Reason for Stop
03 0 03	Power of floating point constant exceeds 50 or is less than 00. (00 > PP > 99)
62 0 73 }	Nest of DO loops exceeding 4.
62 0 99 }	Floating point exponent of a constant.
65 0 99	Subroutine entry exceeds limit.
67 0 99	Unconditional transfer exceeds limit.
69 0 73	Subscripted fixed point variable in error.
69 0 99	Non-subscripted fixed point variable in error.
88 0 73	Subscripted floating point variable in error.
88 0 99	Non-subscripted floating point variable in error.
50 0 50	First character of arithmetic statement or the "DO" statement index is not alphabetic.
xx 0 xx	Errors not specified above will indicate an improper formation of an arithmetic statement.

Error Procedure

When an error occurs during compilation, note the reason for making corrections to the original FORTRAN statements. Depressing the program start key will cause the 650 to read the next statement and continue. Further output of the statement in error will not be punched.

Operating Instructions Phase III

Phase III (Assembly)

As indicated previously, the SOAP II program has been modified for the FOR TRANSIT system. Standard SOAP II decks should not be used in conjunction with the FOR TRANSIT system.

Console Settings

Storage Entry: 70 1952 9999

Switches: Same as for Phase I

Operation

1. Ready machine with proper console settings, FOR TRANSIT 533 control panel and blank cards in the punch hopper.

2. Ready Read Hopper with:

   a. SOAP-PACKAGE Deck
   b. Subroutines in five-per-card absolute format, if any
   c. Entry point "SYN" cards for subroutines in 650 language
   d. Data reservation card (first card of Phase II output)
   e. Subroutines in symbolic SOAP II format, if any
   f. Compiler output in symbolic SOAP II format except first card
   g. One blank card if it is desired to punch out the availability table after assembly. (Availability table can also be obtained by manually transferring control to location 1900)

3. Set the storage entry sign switch to plus for condensed five-per-card output format, or to minus for standard SOAP II format.

4. Press computer reset, program start, and, when the read hopper empties, end-of-file.

5. Run cards out of the punch feed. Discard the first and last cards. With the storage entry sign switch set plus for five-per-card format the remaining cards are:

   1. five-per-card load routine
   2. package subroutines
   3. subroutines entered in five-per-card format
   4. subroutines entered in SOAP II format
   5. compiler output (object program)

   All of the above cards are in the standard five-per-card format. The last card when loaded will also transfer control to location 1999, which is the first location of the object program.

   With the storage entry sign switch set to minus, the remaining cards are:

   1. five-per-card load routine
   2. package subroutines
   3. subroutines entered in five-per-card format

4. one card which modifies the load routine to load single card instructions

All of the above cards are in the standard five-per-card format. The remaining cards are in assembled single card SOAP II format.

5. data reservation card
6. entry point "SYN" cards for subroutines in 650 language
7. subroutines entered in SOAP II format
8. compiler output (object program) assembled in single card SOAP II
9. a transfer card which when loaded will transfer control to location 1999 which is the first location of the object program

Assembling More Than One Program at a Time

Assembly of additional programs requires reloading the SOAP-PACKAGE deck preceding the subroutines and compiler output, since the five-per-card load routine and package subroutines are interspersed throughout the deck and are punched out while the program is loaded. Thus the output from the assembly phase is a complete object program.

Multifile processing can be simulated by inserting a SOAP-PACKAGE deck between each problem to be assembled. The system is such that after processing the last compiled instruction the program transfers control to the console which will load the SOAP-PACKAGE again.

Programmed Stops

The address lights on the console indicate the error condition listed below:

Code	Reason for Stop
0111	Symbol table full.
0222	Drum packed.
0333	Illegal SOAP II card has been encountered. Depress Program Start to continue assembly.

Error Procedure

Error stops 0111 and 0222 are encountered when the capacity of the assembly routine or the drum has been exceeded and assembly is halted.

Error stop 0333 will in the case of single card assembled output reproduce the error card as an input card, and in the case of five-per-card output insert blanks in the instruction and its respective address and the card will not be punched as a load card. Thus, the output cards can be corrected using storage available as indicated from the Availability Table.

SUMMARY OF OPERATING PROCEDURE

CONSOLE SETTINGS

Storage Entry Switches	70 1952 9999	Display	UPPER
Programmed	STOP	Overflow	SENSE
Half Cycle	RUN	Error	STOP
Control	RUN		

If processor program and function title cards are already loaded, set Storage Entry Switches to: 00 0000 1999 or transfer control to location 1999 by means of Address Selection Switches.

OPERATION	PHASE I	PHASE II	PHASE III	OBJECT PROGRAM TIME
1. Insert control panel in 533 for FOR TRANSIT I & II FOR TRANSIT I (S) & II (S)	FOR TRANSIT Phase I board FOR TRANSIT special character board	FOR TRANSIT Phase II & III board FOR TRANSIT special character board	FOR TRANSIT Phase II & III board FOR TRANSIT special character board	FOR TRANSIT Phase II & III board FOR TRANSIT special character board
2. Ready punch hopper with	Blanks	Blanks	Blanks	Blanks
3. Ready read hopper with	a)FOR TRANSIT deck b)Function title cards, if any c)FORTRAN statement cards	a) IT Deck b) IT Header c) IT Statements	a) SOAP-package deck b) Subroutines in 650 language c) Entry point SYN cards for 650 language subroutines d) Data reservation card from Phase II e) Subroutines in SOAP II format f) Compiler output in SOAP II format	a) Output from Phase III, discarding first and last cards b) Data cards, if any
4. Press a) Computer reset key b) Program start key c) End-of-File key to cause:	Translation	Compilation (2)	Assembly	Running of object program
5. At the conclusion of each Phase, run cards out of punch feed and discard first and last card. Remaining cards are:	a) IT statements b) IT header (1)	a) Data reservation card b) Compiled program in SOAP II format (3)	a) Load routine b) Package Subroutines c) Subroutines d) Object program	As asked for by punch statements in object program

1. Rearrange with Header preceding statements.
2. Storage Entry Sign Switch set to minus to eliminate compilation of object program check for underflow or overflow in FOR TRANSIT II & II (S).
3. Insert subroutines in absolute and symbolic SOAP II format to give the order shown as input to Phase III.

CHAPTER IV - USING THE OBJECT PROGRAM

This chapter contains the information and instructions necessary for utilizing an object program produced by the FOR TRANSIT system. The first section deals with the preparation of data cards, and the second section consists of operator's instructions and notes for running the object program.

Preparing Data Cards

As indicated in Chapter I, a READ or PUNCH statement in the source program will cause the object program to read or punch data cards, card after card, until the complete List has been processed. This reading and punching of data is actually accomplished by subroutines contained in the Package Subroutines which are always loaded with the object programs.

Data Cards

Data cards are identified by a "12" punch over card column 73. One to seven pieces of data may be included in one card. If the List requires more than one card, i.e., more than seven pieces of data, additional cards are read or punched.

Data is located in the first seven fields of the card, each field ten columns, with negative values indicated by an "11" punch over the units position of the respective field and positive values by a "12" punch or a blank over the units position.

Word eight of input cards is available for identification as desired by the programmer. Word eight of output cards is punched with the statement number in the I-address, and with a serial number (which is sequential for each problem) in the D-address.

Data is read and punched in the order specified by the LIST from left to right, with arrays in columnwise sequence.

Form of Data

Data representing values of floating point variables are punched in data cards as floating point numbers of the form $\bar{.}$XXXXXXXXPP, where PP is the power of 10 with 50 added to avoid negative exponents. The values assigned to fixed point variables must be integers, and any unused (high order) positions of a field must be punched with zeros.

Executing the Object Program

Operating Instructions

Console Settings

Storage Entry: 70 1952 9999 (If object program already loaded 00 0000 1999)

Switches: Same as for FOR TRANSIT (Phase I)

Operation

1. Ready machine with proper console settings, FOR TRANSIT 533 control panel, and blank cards in the punch hopper.

2. Set storage entry sign switch to minus for execution and to plus for bypassing of conditional Punch Statements.

3. Ready read hopper with entire output of the assembly phase, and data cards if required by the program.

4. Depress computer reset; program start; and when read hopper empties, end-of-file.

The object program will load; control is then transferred to location 1999 which is the first instruction of every object program. Under control of the object program, data cards will be read in and punched out.

Running More Than One Program at a Time

Running more than one object program at a time can be accomplished by stacking the object programs and their respective input data cards in the read hopper. After completing a program the 650 will return control to the console which being set to 70 1952 9999, will load the next object program.

Programmed Stops

PAUSE or STOP statements in the source program will give rise to stop codes in the object program. The data address of the stop instruction in the display lights is available to determine at what point in the program the stop occurred.

Source programs which attempt to transfer control to statement number zero will give rise to an error stop which displays the address 9888 in the display lights. In FOR TRANSIT I (S) and II (S) the number zero is assigned to blank statement numbers, and this number should not be used.

In addition to compiled stops in the object program, certain stops will occur in the package subroutines. The address lights on the console indicate the type of error according to the following lists:

PROGRAMMED STOPS IN PACKAGE DECK FOR FOR TRANSIT I

Address Lights	Error Condition	Package Subroutines in Which Error Can Occur
0001	Negative argument	E00AB, E00L0
0020	Zero argument with negative exponent	E00AK, E00AL
0050	Floating point result less than 10^{-51} or greater than 10^{49}	E00AL, E00LP E00AC, E00AG E00AI, E00AJ E00A0
0501	Floating point number to be fixed greater than 10^{10}	E00TH

Error Procedure

The various error conditions listed above may result from such causes as logical errors or scaling problems inherent in the source program, errors in preparing data cards, etc. Depressing the program start key will cause the 650 to perform the instruction contained in the distributor, which will be the subroutine Exit instruction. This instruction should be noted as an aid in locating that point in the object program where the error occurred.

PROGRAMMED STOPS IN PACKAGE DECK FOR FOR TRANSIT II

Address Lights	Error Condition	Package Subroutines in Which Error Can Occur
0001	Negative or Zero Argument	E00AB, E00LO
0002	Floating Point result $\geq 10^{49}$	E00AC, E00LP

Address Lights	Error Condition	Package Sub-routines in Which Error Can Occur
0003	Error - Floating Point Exponentiation	E00LQ
0010	Fixed Point argument of zero with negative exponent	E00AK
0011	Floating Point argument of zero with negative exponent	E00AL
0049	Floating Point result $\geq 10^{49}$	E00AL
0100	Floating point overflow or underflow in an arithmetic statement	E00AA
0501	Floating Point number to be fixed $\geq 10^{10}$	E00TH

Error Procedure

The various error conditions listed above may result from such causes as logical errors or scaling problems inherent in the source program, errors in preparing data cards, etc. Depressing the program start key will cause the 650 to perform the instruction contained in the distributor, which will be the subroutine Exit instruction. The instruction in the distributor should be noted as an aid in finding the point in the object program where the error was encountered.

APPENDIXES

Appendix A	533 Control Panel Wiring Diagrams for FOR TRANSIT I and FOR TRANSIT II:
	FOR TRANSIT (Phase 1) IT - SOAP (Phases 2 and 3)
	533 Control Panel Wiring Diagram for FOR TRANSIT I(S) and FOR TRANSIT II(S) (all phases).
Appendix B	Listings of Cards for Each Phase of Processing of Sample Problem 4 (Matrix Multiplication).
	Listings of Cards to Illustrate a Function of Multiple Arguments of which One Argument Is Itself a Function of Multiple Arguments.
Appendix C	Glossary.

APPENDIX A

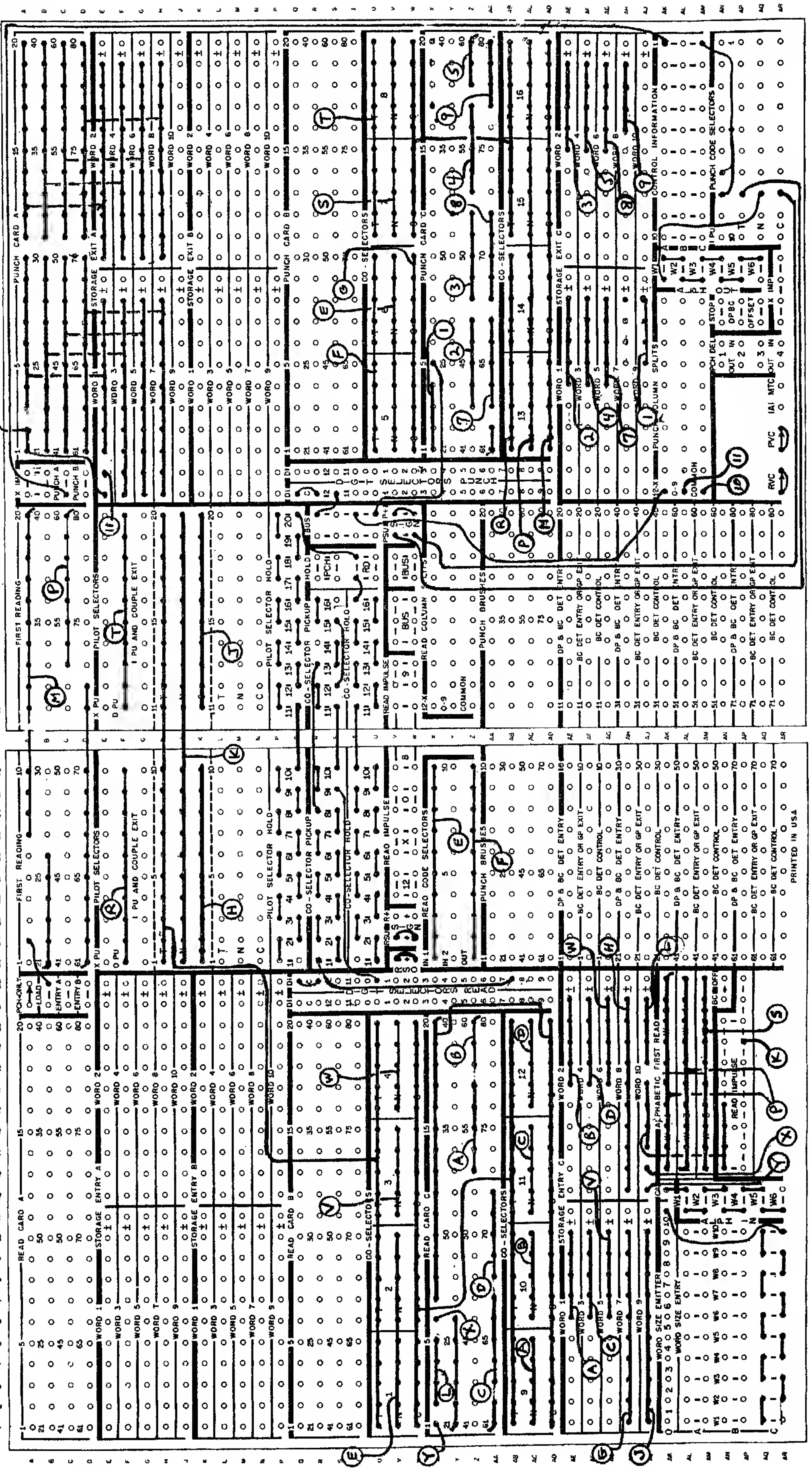
INTERNATIONAL BUSINESS MACHINES CORPORATION

650 DATA-PROCESSING SYSTEM

533-537 CARD READ PUNCH, CONTROL PANEL

FOR TRANSIT I and FOR TRANSIT II

FOR TRANSIT (Phase 1)



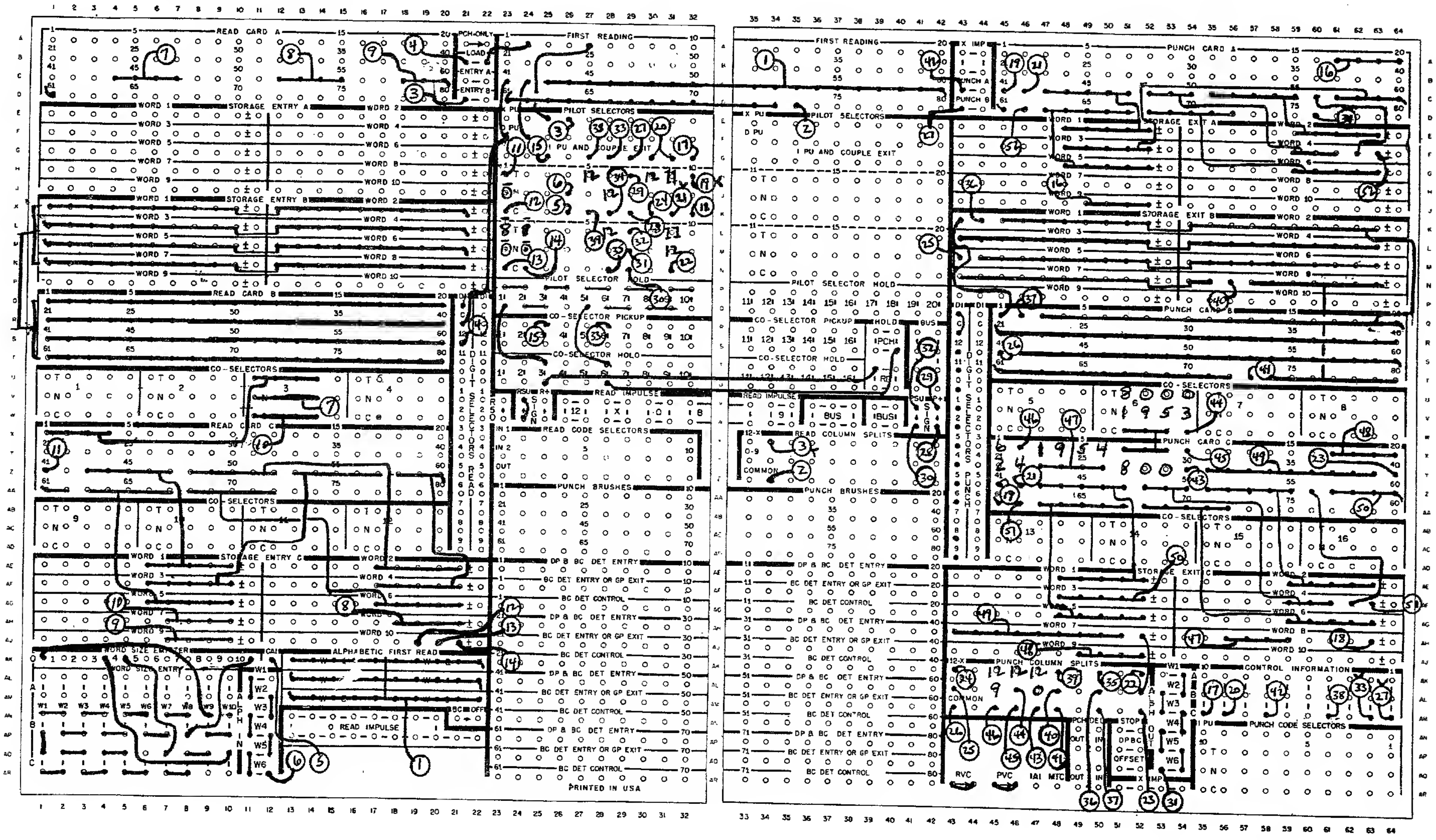
IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

650 DATA-PROCESSING SYSTEM
533-537 CARD READ PUNCH, CONTROL PANEL

FOR TRANSIT I and FOR TRANSIT II

IT-SOAP (Phases 2 and 3)



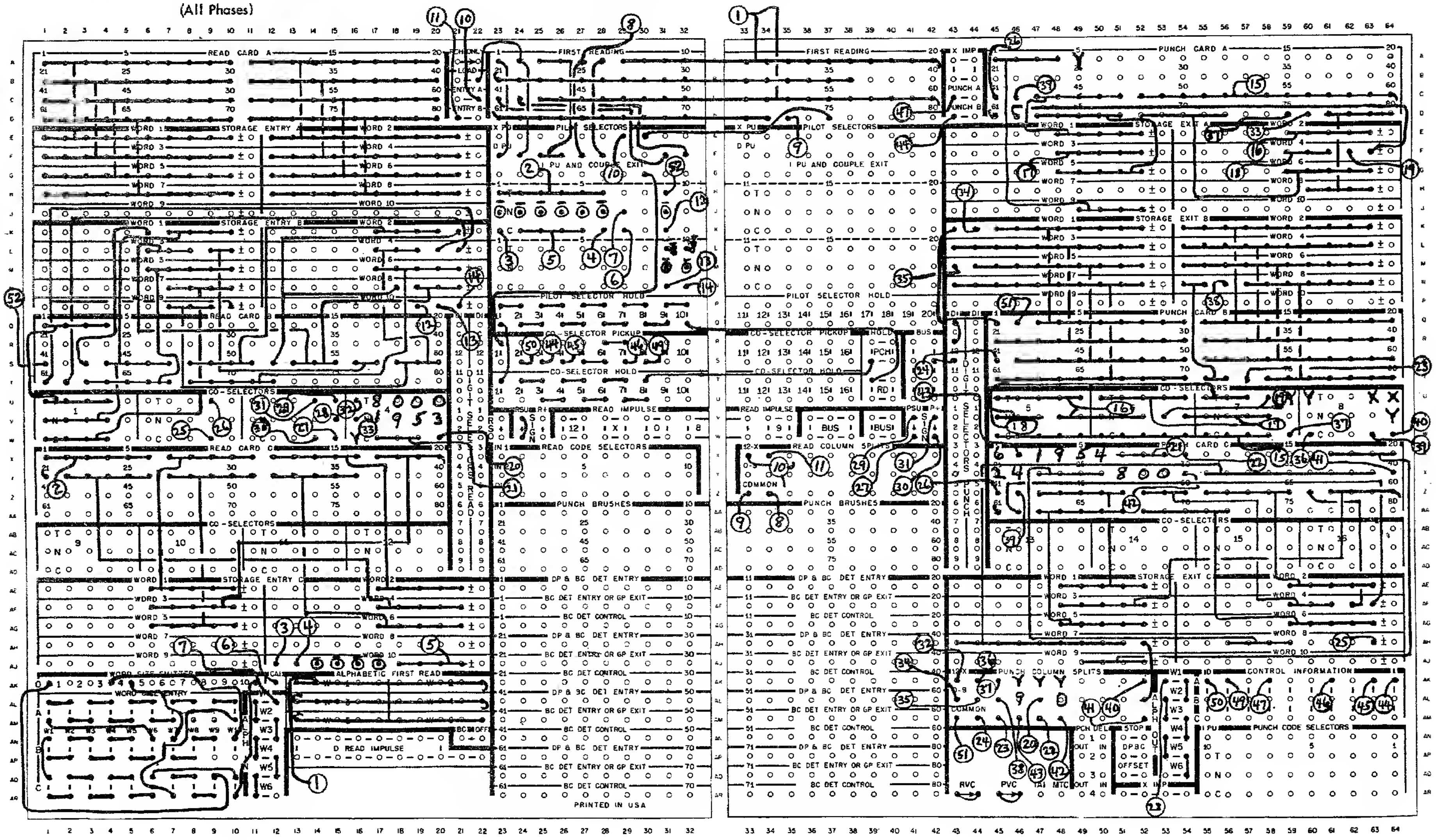
IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

650 DATA-PROCESSING SYSTEM
533-537 CARD READ PUNCH, CONTROL PANEL

FOR TRANSIT I(S) and FOR TRANSIT II(S)

(All Phases)



APPENDIX B - LISTINGS OF CARDS FOR EACH PHASE OF PROCESSING OF SAMPLE PROBLEM 4 - MATRIX MULTIPLICATION

Listing of FOR TRANSIT Source Program Statement Cards for FOR TRANSIT I and FOR TRANSIT II

```
C00000RECTANGULAR MATRIX                                       99999999999999999999
C00000  MULTIPLICATION                                         99999999999999999999
      DIMENSION A(4,5),                                        99999999999999999999
     1  B(5,3)                                                 99999999999999999999
      READ 1,A,B                                               99999999999999999999
      READ 1,N,M,L                                             99999999999999999999
    70DO 4 J=1,N                                               99999999999999999999
    10DO 4 I=1,M                                               99999999999999999999
    60SUM=0.0                                                  99999999999999999999
    20DO 3 K=1,L                                               99999999999999999999
    30SUM = SUM+A(I,K) *                                       99999999999999999999
    31 B(K,J)                                                  99999999999999999999
    40PUNCH 1, SUM, I,J                                        99999999999999999999
    80END                                                      99999999999999999999
```

Listing of FOR TRANSIT Source Program Statement Cards for FOR TRANSIT I(S) and FOR TRANSIT II(S)

```
C      RECTANGULAR  MATRIX
C        MULTIPLICATION
       DIMENSION A(4,5), B(5,3)
       READ 1,A,B
       READ 1,N,M,L
     7 DO 4 J=1,N
     1 DO 4 I=1,M
     6 SUM=0.0
     2 DO 3 K=1,L
     3 SUM = SUM+A(I,K) * B(K,J)
     4 PUNCH 1, SUM, I,J
     8 END
```

NOTE: Sample problem 4 provides a test case for the system. It is recommended that the source program be processed through each phase of the system and the output for each step compared with the appropriate listings in this appendix.

Listing of Output from Translation Phase Consisting of IT Statements and the IT Header Card, Which Has Been Moved to Preceed the IT Statements.

```
                    4B0000000000           +          +        +
                       T200001T15  0021                DF    0000
                       T36T3  7T38                     DF    0000
7+                        4KI39       K    1      K          0007
7+                            1       KI36        K     F    0007
1+                        4KI40       K    1      K          0001
1+                            1       KI37        K     F    0001
6+                     Y41Z0    J0                      F    0006
2+                        3KI42       K    1      K          0002
2+                            1       KI38        K     F    0002
3+                     Y41ZY41SYLLM4RSL4XI42RSI40RXYL        0003
3+                     15SL5XI39RSI42R                  F    0003
4+                     T41T4  0T39                      F    0004
8+                                                     FF    0008
```

Listing of Output from Compilation Phase of
FOR TRANSIT I: The Program in SOAP II Symbolic Form

```
        REG   Y0002    0043
ES000   00     0000   LAAAA
LAAAA   RAL   EZ001              T200001T15
        STL   W0002                  21
        RAL   EZ002                         DF
        STL   W0003
        RAL   EZ003
        LDD   LABAA   E00AQ
ES000   00     0000   LABAA
LABAA   RAL   EZ004              T36T3 7T38
        STL   W0002
        RAL   EZ005                         DF
        STL   W0003
        RAL   EZ006
        STL   W0004
        RAL   EZ007
        LDD   LACAA   E00AQ
FS007   00     0000   LACAA
LACAA   RAL   EZ008              I39   Z
        STL   Y0039   LADAA      1
ET020   00     0000   LADAA
ES001   00     0000   LADAA
LADAA   RAL   EZ008              I40   Z
        STL   Y0040   LAEAA      1
FT024   00     0000   LAEAA
ES006   00     0000   LAEAA
LAEAA   RAL   FZ009              Y41Z0     J0
        STL   Y0041   LAFAA
FS002   00     0000   LAFAA
LAFAA   RAL   EZ008              I42   Z
        STL   Y0042   LAGAA      1
ET031   00     0000   LAGAA
ES003   00     0000   LAGAA
LAGAA   RAU   Y0039              Y41ZY41SYL
        MPY   EZ010              LM4RSL4XI4
        ALO   Y0042              2RSI40RXYL
        SLT   Y0003              15SL5XI39R
        ALO            8002      SI42R
        RAL   Y0015                         F
        STL   W0000
        RAL   Y0040
        STL   W0001
        RAU   Y0042
        MPY   EZ011
        ALO   W0001
        STL   W0001
        RSL   EZ011
        ALO   W0001
        SLT   Y0003
        ALO            8002
        RAL   Y0000
        STL   ACC
        RAL   W0000
        LDD           E00AJ
```

```
        RAL   Y0041
        LDD           E00AI
        STL   Y0041   LAHAA
ES000   00     0000   LAHAA
LAHAA   RAL   EZ008              I42   Z
        ALO   Y0042              I42   S
        STL   Y0042   LAIAA      1
ES000   00     0000   LAIAA
LAIAA   RSL    8002              G       000C
        ALO   Y0038              IF    I38
        BMI   LAJAA   ET031      W     I42
ES004   00     0000   LAJAA
LAJAA   RAL   EZ012              T41T4 0T39
        STL   W0002
        RAL   EZ013                         F
        STL   W0003
        RAL   EZ014
        STL   W0004
        RAL   EZ015
        LDD   LAKAA   E00AR
ES000   00     0000   LAKAA
LAKAA   RAL   EZ008              I40   Z
        ALO   Y0040              I40   S
        STL   Y0040   LALAA      1
ES000   00     0000   LALAA
LALAA   RSL    8002              G       0000
        ALO   Y0037              IF    I37
        BMI   LAMAA   ET024      W     I40
ES000   00     0000   LAMAA
LAMAA   RAL   EZ008              I39   Z
        ALO   Y0039              I39   S
        STL   Y0039   LANAA      1
ES000   00     0000   LANAA
LANAA   RSL    8002              G       0000
        ALO   Y0036              IF    I36
        BMI   LAOAA   ET020      W     I39
ES008   00     0000   LAOAA
LAOAA   NOP    8000    8000
EZ015   00     0003    0004
FZ014   00     0000    0041
EZ013   00     0000    0040
EZ012   00     0000    0039
EZ011   00     0000    0004
EZ010   00     0000    0005
EZ009   00     0000    0000
EZ008   00     0000    0001
FZ007   00     0003    0000
FZ006   00     0000    0036
EZ005   00     0000    0037
EZ004   00     0000    0038
EZ003   00     0002    0000
FZ002   00     0020    0001
FZ001   00     0015    0021
        BOP
```

Listing of Output from Compilation Phase of
FOR TRANSIT II: The Program in SOAP II Symbolic Form

```
        REG   Y0002   0043                  |         SLT   Y0003
ES000   00    0000   LAAAA                  |         ALO           8002
LAAAA   RAL   EZ001          T200001T15     |         RAU   Y0000
        STL   W0002              21         |         FMP   W0000
        RAL   EZ002                  DF     |         FAD   Y0041
        STL   W0003                         |         STU   Y0041   LAHAA
        RAL   EZ003                         | ES000   00     0000   LAHAA
        LDD   LABAA  E00AQ                  | LAHAA   AXC   Y0000
ES000   00    0000   LABAA                  |         RSL    8007
LABAA   RAL   EZ004          T36T3 7T38     |         STD   Y0042
        STL   W0002                         |         ALO   Y0038
        RAL   EZ005                  DF     |         BMI   LAIAA   ET034
        STL   W0003                         | ES004   00     0000   LAIAA
        RAL   EZ006                         | LAIAA   RAL   EZ011            T41T4 0T39
        STL   W0004                         |         STL   W0002
        RAL   EZ007                         |         RAL   EZ012                     F
        LDD   LACAA  E00AQ                  |         STL   W0003
ES007   00    0000   LACAA                  |         RAL   EZ013
LACAA   RAA   Y0000             4KI39       |         STL   W0004
        LDD    8005              K    1     |         RAL   EZ014
        STD   Y0039  LADAA       K          |         LDD   LAJAA   E00AR
ET021   00    0000   LADAA                  | ES000   00     0000   LAJAA
ES001   00    0000   LADAA                  | LAJAA   AXB   Y0000
LADAA   RAB   Y0000             4KI40       |         RSL    8006
        LDD    8006              K    1     |         STD   Y0040
        STD   Y0040  LAEAA       K          |         ALO   Y0037
ET026   00    0000   LAEAA                  |         BMI   LAKAA   ET026
ES006   00    0000   LAEAA                  | ES000   00     0000   LAKAA
LAEAA   RAU   EZ008          Y41Z0     J0   | LAKAA   AXA   Y0000
        STU   Y0041  LAFAA                  |         RSL    8005
ES002   00    0000   LAFAA                  |         STD   Y0039
LAFAA   RAC   Y0000             3KI42       |         ALO   Y0036
        LDD    8007              K    1     |         BMI   LALAA   ET021
        STD   Y0042  LAGAA       K          | ES008   00     0000   LALAA
ET034   00    0000   LAGAA                  | LALAA   NOP    8000    8000
ES003   00    0000   LAGAA                  | EZ014   00     0003    0004
LAGAA   RAU    8005          Y41ZY41SYL     | EZ013   00     0000    0041
        MPY   EZ009          LM4RSL4XI4     | EZ012   00     0000    0040
        SLT   Y0003          2RSI40RXYL     | EZ011   00     0000    0039
        ALO           8002   15SL5XI39R     | EZ010   00     0000    0004
        RAU    6016          SI42R          | EZ009   00     0000    0005
        STU   W0000                   F     | EZ008   00     0000    0000
        RAL    8006                         | EZ007   00     0003    0000
        STL   W0001                         | EZ006   00     0000    0036
        RAU    8007                         | EZ005   00     0000    0037
        MPY   EZ010                         | EZ004   00     0000    0038
        ALO   W0001                         | EZ003   00     0002    0000
        STL   W0001                         | EZ002   00     0020    0001
        RSL   EZ010                         | EZ001   00     0015    0021
        ALO   W0001                                   BOP
```

Listing of Output from SOAP Phase of FOR TRANSIT I: The Object Program in Five-per-Card Form

```
7842220086  0000000000  6500520057  2019680071  6500740079  2019690072  0000199900  5700710079
7842220087  6500750129  6900821913  0000000000  6500850089  2019680121  0072012900  0000820089
7842220088  6501240179  2019690122  6501250229  2019700073  6500760081  0121017901  2202290073
7842220089  6900841913  0000000007  6500870091  2000400093  0001000200  0081000000  8400910000
7842220090  0000000001  6500870141  2000410044  0001000204  0000000006  0000009301  4100000000
7842220091  6500470001  2000420045  0000000002  6500870191  2000430046  0044000100  0000450191
7842220092  0001000301  0000000003  6000400095  1900480068  1500430097  0000000000  4600950068
7842220093  3500040107  1500608002  6500160171  2019660069  6500410145  0097010700  6001710069
7842220094  2019670070  6000430147  1900500120  1519670221  2019670170  0145007001  4701200221
7842220095  6600500055  1519670271  3500040131  1501348002  6500010105  0170005502  7101310134
7842220096  2000000053  6519660321  6901741902  6500420197  6901001852  0105005303  2101740197
7842220097  2000420195  0000000000  6500870241  1500430247  2000430096  0100000001  9502410247
7842220098  0000000000  6680020155  1500390143  4601460046  0000000004  0000009601  5501430000
7842220099  6500490103  2019680371  6502240279  2019690172  6501750329  0146010303  7102790172
7842220100  2019700123  6501260181  6901841907  0000000000  6500870291  0329012301  8100000184
7842220101  1500410245  2000410094  0000000000  6680020153  1500380193  0291024500  0000940153
7842220102  4601960044  0000000000  6500870341  1500400295  2000400243  0193000001  9603410295
7842220103  0000000000  6680020051  1500370391  4601440093  0000000008  0000024300  5103910000
7842220104  0080008000  0000030004  0000000041  0000000040  0000000039  0144012601  7502240049
7842220105  0000000004  0000000005  0000000000  0000000001  0000030000  0050004800  4700870076
7842220106  0000000036  0000000037  0000000038  0000020000  0000200001  0125012400  8500750074
7842220107  0000150021  0000000037  0000000038  0000020000  0000001999  0052000000  0000001980
```

Listing of Output from SOAP Phase of FOR TRANSIT II: The Object Program in Five-per-Card Form

```
7842240078  0000000000  6500520057  2019680071  6500740079  2019690072  0000199900  5700710079
7842240079  6500750129  6900821913  0000000000  6500850089  2019680121  0072012900  0000820089
7842240080  6501240179  2019690122  6501250229  2019700073  6500760081  0121017901  2202290073
7842240081  6900841913  0000000007  8000010090  6980050046  2400400093  0081000000  8400900046
7842240082  0001000201  0000000001  8200010049  6980060055  2400410044  0000000000  9300490055
7842240083  0001000206  0000000006  6000470001  2100420045  0000000002  0000000000  4400010000
7842240084  8800010051  6980070107  2400430096  0001000304  0000000003  0045005101  0700000000
7842240085  6080050053  1900560126  3500040087  1501408002  6060160171  0096005301  2600870140
7842240086  2119660069  6580060077  2019670070  6080070127  1900800050  0171006900  7700700127
7842240087  1519670221  2019670120  6600800135  1519670271  3500040131  0050022101  2001350271
7842240088  1501348002  6000010105  3919660066  3200420119  2100420095  0131013401  0500660119
7842240089  0000000000  5800010101  6680070059  2400430146  1500390143  0000009501  0100590146
7842240090  4601960096  0000000004  6500990103  2019680321  6501740279  0143000001  9601030321
7842240091  2019690172  6501750329  2019700123  6501760181  6901841907  0279017203  2901230181
7842240092  0000000000  5200010190  6680060097  2400410094  1500380193  0000018401  9000970094
7842240093  4602460044  0000000000  5000010102  6680050109  2400400243  0193000002  4601020109
7842240094  1500370091  4601440093  0000000008  0080008000  0000030004  0243009100  0001440176
7842240095  0000000041  0000000040  0000000039  0000000004  0000000005  0175017400  9900800056
7842240096  0000000000  0000030000  0000000036  0000000037  0000000038  0047007601  2501240085
7842240097  0000020000  0000200001  0000150021  0000000037  0000001999  0075007400  5200001980
```

Note: The above listings do not include the SOAP-PACKAGE cards produced by the SOAP Phase. Accordingly, the card serial numbers (word 1, columns 8-10) begin at 86 and 78, respectively.

Actual problem and answer matrix for Input and Output data used with sample problem 4.

$$\begin{bmatrix} 16 & 13- & 7- & 2 & 5 \\ 2 & 1 & 8 & 10- & 12- \\ 1 & 6- & 15 & 11 & 18 \\ 14 & 17 & 3 & 2- & 9 \end{bmatrix} \times \begin{bmatrix} 3- & 5 & 7 \\ 8- & 13 & 4- \\ 6 & 4- & 10 \\ 12 & 3 & 5- \\ 2 & 9- & 11 \end{bmatrix} = \begin{bmatrix} 48 & 100- & 139 \\ 110- & 69 & 8 \\ 303 & 262- & 324 \\ 166- & 192 & 169 \end{bmatrix}$$

Listing of Input Data Cards for Object Program

```
1600000052   2000000051   1000000051   1400000052   1300000052-  1000000051   6000000051-  0000000002
1700000052   7000000051-  8000000051   1500000052   3000000051   2000000051   1000000052-  0000000003
1100000052   2000000051-  5000000051   1200000052-  1800000052   9000000051   3000000051-  0000000004
8000000051-  6000000051   1200000052   2000000051   5000000051   1300000052   4000000051-  0000000005
3000000051   9000000051-  7000000051   4000000051-  1000000052   5000000051-  1100000052   0000000006
0000000003   0000000004   0000000005                                                      0000000001
```

Listing of Output (Answer) Cards from Object Program

```
4800000052  0000000001  0000000001  0000000000  0000000000  0000000000  0000000000  0000010004
1100000053- 0000000002  0000000001  0000000000  0000000000  0000000000  0000000000  0000020004
3030000053  0000000003  0000000001  0000000000  0000000000  0000000000  0000000000  0000030004
1660000053- 0000000004  0000000001  0000000000  0000000000  0000000000  0000000000  0000040004
1000000053- 0000000001  0000000002  0000000000  0000000000  0000000000  0000000000  0000050004
6900000052  0000000002  0000000002  0000000000  0000000000  0000000000  0000000000  0000060004
2620000053- 0000000003  0000000002  0000000000  0000000000  0000000000  0000000000  0000070004
1920000053  0000000004  0000000002  0000000000  0000000000  0000000000  0000000000  0000080004
1390000053  0000000001  0000000003  0000000000  0000000000  0000000000  0000000000  0000090004
8000000051  0000000002  0000000003  0000000000  0000000000  0000000000  0000000000  0000100004
3240000053  0000000003  0000000003  0000000000  0000000000  0000000000  0000000000  0000110004
1690000053  0000000004  0000000003  0000000000  0000000000  0000000000  0000000000  0000120004
```

LISTINGS OF CARDS TO ILLUSTRATE FUNCTIONS OF MULTIPLE ARGUMENTS.

Listing of FOR TRANSIT Source Program Statement Cards

```
 Y=OUTF(A,B,C,D,ENRF(                                   9999999999999999999
1E,F,G),H)                                              9999999999999999999
 END                                                    9999999999999999999
```

Listing of IT Statements, Output from Translation Phase.

```
I000000000+          +          +          +
   Y1ZQ16EKY2KY3KY4KY5KQ13EKY6KY7
   KY8QK   Y9Q                    F
                                 FF
```

Listing of Output from Compilation Phase
of FOR TRANSIT I:
The Program in SOAP II Symbolic Form

```
          REG   Y0002   0010
  ES000   00    0000    LAAAA
  LAAAA   RAL   Y0009               Y1ZQ16EKY2
          STL   P0000               KY3KY4KY5K
          RAL   Y0008               Q13EKY6KY7
          STL   P0001               KY8QK   Y9Q
          RAL   Y0007
          STL   P0002                        F
          RAL   Y0006
          LDD           E00AN
          NOP   P0002
          STL   P0001
          RAL   Y0005
          STL   P0002
          RAL   Y0004
          STL   P0003
          RAL   Y0003
          STL   P0004
          RAL   Y0002
          LDD           E00AQ
          NOP   P0004
          STL   Y0001   LABAA
  FS000   00    0000    LABAA
  LABAA   NOP   8000    8000
          BOP
```

Listing of Output from Compilation Phase
of FOR TRANSIT II:
The Program in SOAP II Symbolic Form

```
          REG   Y0002   0010
  ES000   00    0000    LAAAA
  LAAAA   RAU   Y0009               Y1ZQ16EKY2
          STU   P0000               KY3KY4KY5K
          RAU   Y0008               Q13EKY6KY7
          STU   P0001               KY8QK   Y9Q
          RAU   Y0007
          STU   P0002                        F
          RAU   Y0006
          LDD           E00AN
          NOP   P0002
          STU   P0001
          RAU   Y0005
          STU   P0002
          RAU   Y0004
          STU   P0003
          RAU   Y0003
          STU   P0004
          RAU   Y0002
          LDD           E00AQ
          NOP   P0004
          LDD           E00AA
          STU   Y0001   LABAA
  ES000   00    0000    LABAA
  LARAA   NOP   8000    8000
          BOP
```

APPENDIX C

Glossary

FORTRAN System - An automatic coding system originally designed for the IBM 704, intended primarily for scientific computation.

FORTRAN Program - A computer program written in the FORTRAN language.

FORTRAN Language - Statements closely resembling the language of mathematics which are acceptable to a computer as a source program.

FOR TRANSIT System - An automatic coding system for the IBM 650 which uses the FORTRAN language for its source programs and gives optimized machine language programs as output.

Source Program - The input to an automatic coding system. In the FOR TRANSIT system the source program consists of FORTRAN statements.

Object Program - The machine language program which is the final output of an automatic coding system.

Compile - Create a series of sequential machine instructions for actual operation by processing source program statements.

Assemble - Assign actual machine language addresses and operation codes to symbolic addresses and operation codes.

Optimize - Select memory locations which have minimum access time for each operation.

FOR TRANSIT Deck - Cards containing instructions and tables for the 650 processor program which translates FORTRAN statements into IT statements.

IT Deck - Cards containing instructions and tables for the 650 processor program which compiles symbolic machine language instructions from the IT statements.

SOAP II Deck - Cards containing instructions and tables for the 650 program which assembles and optimizes the output of the IT deck to create a machine language object program.

Abbreviations and Acronyms

FORTRAN	FORmula TRANslator
FOR TRANSIT	FORtran TRANSlation to IT, or FORTRAN, SOAP, IT
IT	Internal Translator
SOAP	Symbolic Optimal Assembly Program